INSA LYON

DETAILED RESEARCH REPORT

Analyzing robot Navigation Among Movable Obstacles in unknown environments to extend it to social and dynamic constraints

Author:
Benoit RENAULT

Supervisors:
Fabrice JUMEL
Jacques SARAYDARYAN
Olivier SIMONIN

Examiners:
Jean-François BOULICAUT
Vasile-Marian SCUTURICI
Christine SOLNON

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

A report written in the INRIA Chroma team
and submitted in fulfillment of the requirements of the Engineering Degree
of the Computer Sciences and Engineering Department (IF)

August 22, 2018

Résumé: Nous présentons un état de l'art détaillé du domaine de la NAMO ("Navigation Among Movable Obstacles": Navigation parmi des obstacles mobiles), ainsi qu'un crible de critères de comparaison nous permettant de situer les propositions existantes et la notre. Nous analysons les algorithmes locallement optimaux de NAMO en milieu inconnu, les reformulons en pseudocode et construisons un nouvelle ensemble de propositions sur cette fondation. Nos propositions 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-clefs: NAMO, Navigation parmi des obstacles mobiles, Optimalité, Navigation Sociale, Environnement dynamique

Abstract: We present a detailed state of the art of the NAMO (Navigation Among Movable Obstacles) domain, and a set of comparison criterias to sort through existing propositions and situate our own. We 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. Our propositions are the first required steps toward bringing dynamic and social constraints to the domain. We partially validate our proposition with a ROS-compatible (Robot Operating System) simulator and pushing tests with real Pepper robot, standard platform the Robocup@Home challenge.

Key-words: NAMO, Navigation Among Movable Obstacles, Optimality, Social Navigation, Dynamic environments

Contents

Chapter 1

Introduction

1.1 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, humans (or other autonomous actors, such as pets) are 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 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.

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 as described above, this is not sufficient.

(A) Sample social navigation problem

(B) Sample NAMO problem

FIGURE 1.1: 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 [27]). In the NAMO problem, the robot will need to move obstacles in order to access its goal (image from [33]).

To our knowledge, the problem of navigation planning in dynamic environments populated with humans [14, 26], and the problem of navigation planning in robot-alterable environments (which is called Navigation Among Movable Obstacles, or NAMO, more precisely defined in

[31]) 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 [8, 28], respecting social conventions such as avoiding human interaction spaces [24, 27] (Figure 1.1a), or predicting the trajectories of moving obstacles [10], ... The NAMO problem (Figure 1.1b) has been dealt with in a great variety of contexts, detailed in Chapter 2. 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.

1.2 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 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.

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

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

[3] Pepper characteristics are described in Chapter 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/

1.3 Overview

The following work is organized as follows:

- Chapter 2 is a detailed state of the art of the NAMO domain, and derives comparison criteria from a selection of articles that are closely related to our goals. It also explains the choice of the papers we chose to build upon.
- Chapter 3 is a thorough study and criticism of the chosen base algorithm. We explain the logic of the original algorithm while also providing definitions and conventions that remove the many original ambiguities. Finally, we propose a pseudocode interpretation of the improvements proposed by Levihn et. al. on the first algorithm.
- Chapter 4 revisits the algorithm to really restore optimality, make it stick to our hypotheses, and extend it to solve new social and dynamic environment concerns.
- Chapter 5 recounts our experimentations with the Pepper robot and simulations to validate our propositions.
- Chapter 6 summarizes our contributions and details opportunities for research arisen by this work.
- Appendices A and B gather comparison tables and pseudocode representations of algorithm propositions.

Chapter 2

Navigation Among Movable Obstacles: state of the art

2.1 Determining appropriate comparison criteria

The definition of the scope of NAMO issues has evolved as publications about it became more numerous and dealt with a greater variety of contexts. As a synthesis this diversity, we will summarize it as the problem of having an autonomous agent navigate 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), or to reach it at all if no path exists that allows to reach it without moving obstacles. The many approaches to the problem and their contexts highlight a need to characterize them, thus our establishing comparison criteria.

These comparison criteria are divided into three main categories: hypotheses, approaches and performance criteria. 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 it.

2.1.1 Hypotheses

- **Knowledge of the environment** The most significant criterion to understand the context of a navigation algorithm is "what does the robot know about the environment?". This includes the way it represents it (metric/topologic map, dimensions, ...) and how much the robot knows of it at the moment of the algorithm's execution (none, partial or complete knowledge). If the knowledge is not complete, this criterion also includes hypotheses on the unknown environment (e.g., can the robot consider it free space, or obstacle-occupied space?). All the papers presented in the following pages provide this information.

- **Obstacle characteristics** The nature, shape, cinematic or frictional constraints, associated semantics (physical characteristics like center of gravity, weight, moment of inertia, ...), autonomous movement or not of the considered obstacles all condition the final algorithm proposition, because these characteristics partly define the problem solution space (e.g., poses where to put an object may differ whether we represent it as a rectangle or a polygon, cinematic or frictional constraints may reduce the manipulation possibilities, ...).

- **Robot characteristics** Naturally, in the same way an algorithm depends on the suppositions we make on obstacles, it also depends on the ones we make on the robot itself. When a specific robot is used, it is important to know its manipulation capabilities (only pushes? translations only? rotation?), its geometric representation, its navigation capabilities (differential drive that only allows translation XOR rotation? omnidirectional drive that allows

both at the same time?), but also its sensing capabilities (on-board sensors like cameras or sonars, and their effective range). By comparing the requirements for the robot in a proposition, we can know if it will be applicable with our Pepper robot or not.

- **Problem class** In his first paper [33] published in 2005, Dr.Stilman, the founder of the NAMO domain, established a NAMO problem classification inspired from an existing one in the field of rearrangement planning [2]. The *LP* notation is for "Linear Problem". A NAMO problem has a linear solution when there exists a sequence of free space components such that merging two adjacent components does not constrain the merge of the following adjacent components. The k notation in LP_k relates to the maximal number of obstacles the algorithm should consider when trying to independently connect two disconnected components of free-space. Independently means that the algorithm operates under the assumption that moving k obstacles to create this connection will not affect the solvability of following solutions. k therefore characterizes the exponential complexity of a NAMO problem, since it means that we consider every manipulation combinations for k obstacles when evaluating a connection plan. For example, a situation where two adjacent rooms are linked together through a single door, and a single obstacle is present in the doorway, then the problem class of navigating to the other room is LP_1. If it were necessary to move two obstacles from the doorway to go to the other room, the problem class would be LP_2. The notation is further explained and refined in another paper [32] published in 2008. This notation allows us to quickly know what kind of situation an algorithm will be able to solve or not.

2.1.2 Approaches

- **Path Planning Algorithm(s) and heuristics** The NAMO algorithms in the considered papers are systematically building upon existing graph search algorithms like A* or Dijkstra [1], or use them as path finding subroutines. Often, as is the case when using A*, heuristics are used to direct the search and obtain gains in terms computer time performance. Knowing on which algorithm and heuristics the new proposition is built upon is essential to characterize it, and judge whether it is appropriate for our own proposition.

- **Evaluation and evolution of an obstacle's "movable" characteristic and its associated cost** The way the different algorithms take movable obstacles into account differs from one algorithm to the other. Some propositions depend on knowing beforehand whether an obstacle is actually tagged as movable or not, while others deduce this status on-line (i.e while navigating). Some use a constant cost for manipulating all obstacles, while others compute a cost according to known physical properties of the obstacle, like its weight. Knowing what a proposition is using to compute the cost of manipulating an obstacle is essential to understand its limitations.

- **Object manipulation maneuver planning** The way the proposition has the robot place itself by the obstacle for manipulation matters too. Whether it is only based on geometric considerations or cinematic ones on the obstacle or the robot. Whether the algorithm is or is not capable of reconsidering its placement next to the obstacle depending on new geometric or cinematic information that is collected during the navigation execution also affects the algorithm proposition.

- **Planning taking uncertainty into account** A real world setting, using a real robot, implies limited sensing and actuation capabilities, and therefore, uncertainty about the state of the

[1] Basic presentation http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

environment and the robot. Since our long-term goal is to apply our proposition to real-world situations, it is important to take note of the many different strategies used to take uncertainty into account.

2.1.3 Performance criteria

- **Evaluation in a simulated/real setting** In robotics, it is paramount to know whether an algorithmic proposition has made it to the stage of real-world experimentation or not. Since it is a field that is heavily anchored in reality, an experimental validation in a real-world setting is far more valuable than a simulation.
- **Computation time** In robotics, algorithms that can be executed in a short time span are very valuable, since it is a world of real-time computation and actuation. If computing a new plan were to take more time than it would to execute it, it would arguably be considered too expensive. Thus, knowing if a proposition is executable in real-time or not matters a lot.
- **Optimality and completeness** For about any algorithm, knowing whether it is complete (always return a solution when there is one) and guaranteed to be optimal (the returned solution is always the best) matters. In a context where the robot must make decisions upon partial knowledge on the environment, an algorithm is considered *locally optimal* if it is guaranteed to return the best solution given its current belief state of the environment.
- **Optimality target** Even if an algorithm can not guarantee optimality, it will still try to optimize a certain cost. The type of cost an algorithm can manage is also a significant criterion, since in the context of robotics many types of cost are used: distance, time, energy, risk, ...
- **Social acceptability** This criterion does not come from the reading of our corpus but from the goal we have set for ourselves to seek if there are any social acceptability considerations in existing NAMO algorithms. It is thus interesting to our research to see to what extent the existing propositions care for social problematics or not.
- **Number and Density of obstacles** Since NAMO computations depend on each and every individual obstacle, it makes sense that the number and repartition of obstacles in the environement would affect the computation efficiency of an algorithm. Knowing the maximum values with which algorithms have been successfully tested in real-time conditions, for example, would allow us to compare the efficiency of the different algorithms.

2.1.4 Recapitulative tables

The following tables recapitulate the above-mentioned criteria.

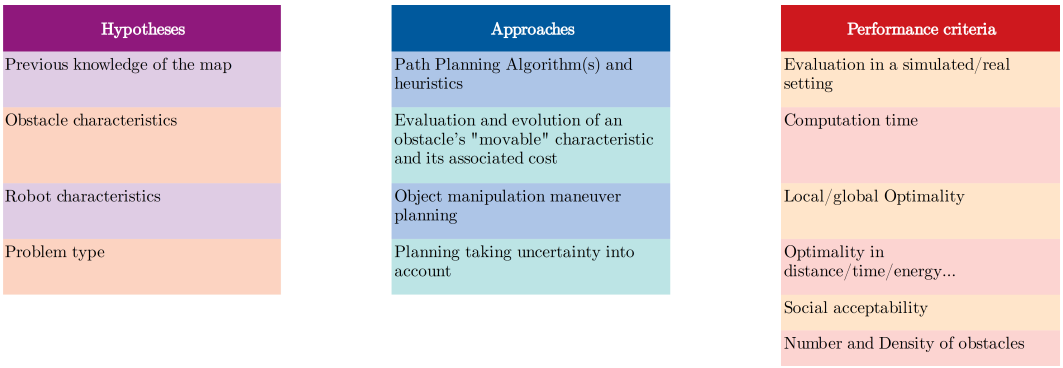


FIGURE 2.1: Comparison criteria, sorted by type: hypotheses, approaches and performance criterias

2.2 Comparison and cross-comparison

2.2.1 Comparison Tables

In order to be able to properly and graphically situate our work in the context of the currently available research in the NAMO domain, we have made several comparison tables according to the criteria presented beforehand. In the following pages, you will find the booleanized comparison tables, where each global criteria is divided into boolean sub-criteria that the paper answers to or not. The detailed comparison and cross-comparison tables are available in the Comparison tables Appendix. Abbreviations are explained in the detailed comparison table.

Emphasized and bold cells with the "[Prop]" reference are criteria that are validated by the final proposition we make in Chapter 4.

Since these table have been made in another software than the one used to write this report, in order to avoid endless export procedures, articles are refered by letters. The table below gives the correspondence with the numbered references of the present document.

Table Reference	[a]	[b]	[c]	[d]	[e]	[f]	[g]	[h]	[i]	[j]	[k]	[l]
Bibliography Reference	[22]	[33]	[34]	[32]	[38]	[12]	[18]	[15]	[16]	[6]	[7]	[30]

FIGURE 2.2: Hypotheses comparison tables

Path Planning Algorithm(s) and heuristics							
A*	ARA*	D* Lite	BFS	RRT	Standard Heuristic for Path Planning	Custom Heuristic for Path Planning	Supplementary Heuristics
[b], [c], [d], [e], [Exp]	[j]	[i], [k]	[d]	[f], [g], [h], [l]	*[b], [c], [e], [i], [j], [Exp]*	[d]	*[b], [c], [d], [e], [g], [i], [j], [k], [l], [Exp]*

Evaluation and evolution of an obstacle's "movable" characteristic and its associated cost				
"Movability" (re)evaluated on runtime	Manipulation cost depends on the obstacle's physics metadata	Manipulation cost depends on a constant common to all obstacles	Cost is estimated on runtime	Cost is pre-estimated by a heuristic
[e], [f], [g], [h], [i], [j], [k], [l], [Exp]	[a], [b]. [c]	*[e], [i], [Exp]*	[j], [k]	[i]

Object manipulation maneuver planning			Planning taking uncertainty into account					
Kinematic/Friction constraints taken into account	Limited grasping points number	No concern about grasping points	Adaptive obstacle approach procedures	Use of a Kalman filter	Use of e-shadows	Use of PRM + MDP + MonteCarlo	Use of PBRL	Pointcloud correction
[c], [h], [l]	*[a], [b], [c], [d], [e], [f], [g], [h], [l], [Exp]*	[j], [k]	[c], [g], [h], [j], [k], [l]	[g], [j], [k]	[g]	[h], [l]	[l]	[f]

FIGURE 2.3: Approaches comparison tables

Optimality type				Social acceptability		Number and Density of obstacles		
Energy optimality	Distance optimality	Time optimality	Other optimality	Mention of social norms/concerns	Takes social norms into account	Maximal tested quantity of "movable obstacles" >= 20	Maximal tested quantity of "movable obstacles" < 20	Mention of the concept of obstacle density
[a], [b], [c], [e], [h], [i], [l]	*[a], [d], [f], [g], [k], [Exp]*	[h], [j], [k], [l]	*[b], [d], [f], [g], [h], [k], [Exp]*	*[b], [f], [Exp]*	*[Exp]*	[b], [e], [h], [i]	*[a], [c], [d], [f], [g], [j], [k], [l], [Exp]*	[e], [i]

Evaluation in a simulated/real setting		Computation time	Optimality and completeness		
Evaluation in a real-world setting	Evaluation in a simulation	Real time	Guaranteed Global Optimality	Guaranteed Local Optimality	Guaranteed Completeness
[c], [f], [g], [j], [k], [l]	*[a], [b], [d], [e], [h], [i], [Exp]*	*[b], [c], [d], [e], [f], [g], [i], [j], [k], [i], [Exp]*	[b], [h]	*[i], [Exp]*	[b], [c], [h]

FIGURE 2.4: Performance criteria comparison tables

2.2.2 Analysis

The first remark we can draw 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, in Figure 2.2, the Obstacle characteristics and Robot characteristics tables show 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 [33].

As many of the papers [33, 34, 32, 38, 15, 16, 30] 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 [37]. 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 [2]), 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" [32]. Further discussion on the complexity of the domain with illustrations can be found in Stilman's thesis on NAMO [31].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 [22], 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, this is the only paper among all the selected ones 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 [34], the robot might lose its balance.

Figure 2.2 (Robot Characteristics table) also shows that a diversity of real and simulated robots have been used for experimenting with NAMO algorithms, but it is to noted that the ones that were actually used for a real world experiment are very costly robots (PR2 [3], HRP2 [4] and GOLEM [5]) at the exception of the custom robotic platform built by Clingerman [6]. 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.

Figure 2.3 (Path Planning Algorithm(s) and heuristics Table) confirms 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 [7], since it allows him to potentially manage autonomously moving obstacles, since it is based off the D*Lite algorithm (see [13] and [6]), 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 [33, 38]. Some papers [34, 18, 15, 6, 7, 30] 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 [15, 30].

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-time performance is paramount, and trading off optimality for performance is often prefered. However, sensible approaches like [33, 38], where the proposition contains an original algorithm that is optimal, but also a modified version that improves performance at the cost of

[2] https://en.wikipedia.org/wiki/Sokoban
[3] Characteristics: http://www.willowgarage.com/pages/pr2/specs
[4] Characteristics: http://global.kawada.jp/mechatronics/hrp2.html
[5] Characteristics: http://www.golems.org/projects/krang.html
[6] http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf

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 [5, 23, 21, 3, 11, 19, 17] 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. [12] that brought the first extension to NAMO in unknown environments [16] 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 [38, 16]. 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 [18]. Clingerman also proposed later another solution for NAMO in unknown environments [6, 7], 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, it is to be noted that none of the selected papers (and also the ones that were only briefly examined) consider a case with movable obstacles mixed with humans (Figure 2.2, Obstacle characteristics). Also, if Stilman [33] and Kakiuchi [12] 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.

2.2.3 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 [38, 16], 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 [29]), and the grasping capabilities of Pepper are very limited (low applied force in particular).

Also, it is noteworthy that most of Wu's proposition was actually formulated with pseudocode, making it easier to build upon. This is not an "official" criterion, but given the timeframe available for this work, no time could have been afforded for guess-work as to how a proposition really works. The next chapter will therefore thoroughly present the solution proposed by Wu et. al. [38], and apply the improvements proposed by [16]. Then, we will be able to bring our own propositions.

Chapter 3

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

3.1 Original algorithms

In Wu et. al.'s proposition, 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 Figure 3.1, 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).

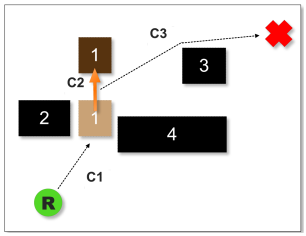


FIGURE 3.1: Figure describing the three plan components when pushing an object, as proposed in [38].

In their first article [38], Wu et. al. describe two versions of their algorithm: a naive but locally optimal baseline one, and an optimized one, built upon the logic foundation of the first, but that loses its local optimality. Several other improvements on performance are given that restore the local optimality of the original proposition in their second article [16]. As a reminder, by locally optimal, we mean that the algorithm will **always** choose the **best** plan given its current, limited knowledge of the environment.

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[1] for A* and the opening detection algorithm (presented in the Third Optimization paragraph). Whereas the first proposition is limited to rectangular obstacles, the second is extended to any polygonal obstacles, and none take autonomously moving obstacles 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,

[1]Occupancy grid defition: https://www.mathworks.com/help/robotics/ug/occupancy-grids.html

but can also pull them in the second one. The NAMO class of the problem is a subset of L_1, since a given plan can only ever imply the movement of one obstacle.

BASELINE(R_{Init}, R_{Goal})

$R \Leftarrow R_{Init}$;
$\mathcal{O} \Leftarrow \emptyset$; {set of objects}
$p_{opt} \Leftarrow A^*(R_{Init}, R_{Goal})$;
while $R \neq R_{Goal}$ **do**
  $\mathcal{O}_{new} \Leftarrow$GET-NEW-INFORMATION();
  if $\mathcal{O}_{new} \neq \emptyset$ **then**
    $\mathcal{O} = \mathcal{O} \cup \mathcal{O}_{new}$;
    for each $o \in \mathcal{O}$ **do**
      for each possible push direction d on o **do**
        $p \Leftarrow$ EVALUATE-ACTION(o,d);
        if $p.cost < p_{opt}.cost$ **then**
          $p_{opt} = p$;
        end if
      end for
    end for
  end if
  $R \Leftarrow$ Next step in p_{opt};
end while

(A) Main loop that evaluates all plans containing the manipulation of an obstacle every time a new one is found

EVALUATE-ACTION(o, d)

$p_{o,d} \Leftarrow \emptyset$
$c_1 = |A^*(R, o.init)|$;
$o.position = o.init$;
while push on o in d possible **do**
  $o.postion = o.postion + one\_push\_in\_d$;
  $c_2 = (o.postion - o.init)$;
  $c_3 = |A^*(o.position, R_{Goal})|$;
  $p = c_1 + c_2 + c_3$;
  $p.cost = c_1 * moveCost + c_2 * pushCost + c_3 * moveCost$;
  $P_{o,d} \Leftarrow P_{o,d} \cup \{p\}$;
end while
return $p \in P_{o,d}$ with min $p.cost$;

(B) Subroutine for evaluating all possible plans for each manipulation direction allowed on an obstacle

FIGURE 3.2: Copy of the Baseline Algorithm proposed by Wu et. al. in [38]. The notations used are not defined in the article, hence our next section 3.2, where detailed definitions can be found.

OPTIMIZED(R_{Init}, R_{Goal})

$R \Leftarrow R_{Init}$;
$P_{sort} \Leftarrow \emptyset$; {list of plans, sorted ascending by $minCost$}
$p_{opt} \Leftarrow A^*(R_{Init}, R_{Goal})$;
while $R \neq R_{Goal}$ **do**
  $\mathcal{O}_{new} \Leftarrow \mathcal{O}_{new} \cup$ GET-NEW-INFORMATION();
  if $p_{opt} \cap \mathcal{O}_{new} \neq \emptyset$ **then**
    $p_{opt} \Leftarrow A^*(R, R_{Goal})$;
    for each $o \in \mathcal{O}_{new}$ **do**
      for each possible push direction d on o **do**
        P_{sort}.insert(OPT-EVALUATE-ACTION(o,d, p_{opt}));
      end for
    end for
    $p_{next} = P_{sort}[0]$;
    while $p_{opt}.cost \geq p_{next}.minCost$ **do**
      p=OPT-EVALUATE-ACTION($p_{next}.o$,$p_{next}.d$, P_{opt});
      if $p.cost < p_{opt}.cost$ **then**
        $p_{opt} = p$;
      end if
      $p_{next} = P_{sort}$.getNext();
    end while
    $\mathcal{O}_{new} \Leftarrow \emptyset$;
  end if
  $R \Leftarrow$ Next step in p_{opt};
end while

(A) Main loop

OPT-EVALUATE-ACTION(o, d, p_{opt})

$P_{o,d} \Leftarrow \emptyset$;
$c_1 = |A^*(R, o.init)|$;
$c_2 = 0$;
$o.position = o.init$;
while push on o in d possible AND $c_2 * pushCost < p_{opt}.cost$ **do**
  $o.postion = o.postion + one\_push\_in\_d$;
  if push created new opening **then**
    $c_2 = (o.position - o.init)$;
    $c_3 = |A^*(o.position, R_{Goal})|$;
    $p = c_1 + c_2 + c_3$;
    $p.cost = c_1 * moveCost + c_2 * pushCost + c_3 * moveCost$;
    $p.minCost = c_2 * pushCost + c_3 * moveCost$;
    $p.o = o$;
    $p.d = d$;
    $P_{o,d} \Leftarrow P_{o,d} \cup \{p\}$;
  end if
end while
return $p \in P_{o,d}$ with min $p.cost$;

(B) Subroutine

FIGURE 3.3: Copy of the Optimized Algorithm proposed by Wu et. al. in [38].

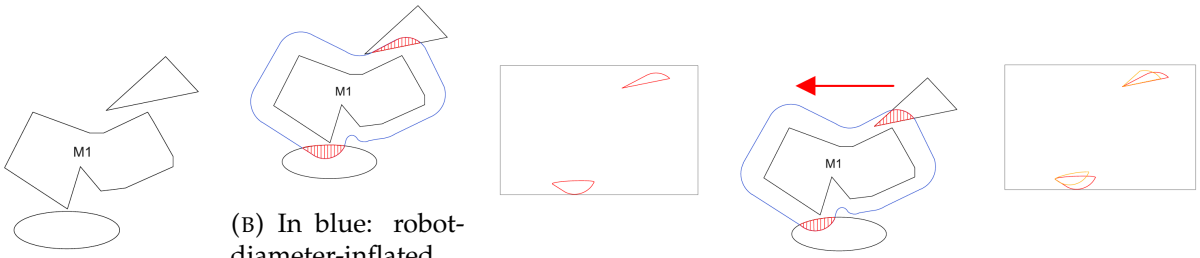
The baseline algorithm is very straightforward: it uses an A* path finding subroutine to determine the optimal path between the current robot's position and the goal, avoiding all known obstacles (none at the beginning). As the robot moves forward (Figure 3.2a, line 17), and therefore gains new information (same figure, line 5), whenever a new obstacle is encountered, every push action for every obstacle is re-simulated (Figure 3.2b) and compared to the current optimal plan (Figure 3.2a, line 11). Local optimality is guaranteed for this approach, since the A* algorithm is used with the admissible Euclidean heuristic, thus returning optimal solutions for a given state of the map and goal, and also because whenever a new obstacle is detected (= the map is in a new state), **all** possible plans are re-evaluated and compared to check if a better plan than the current one can be found or not before moving the robot again.

The optimized version of the algorithm published in the first article offers four optimization steps:

First Optimization Only consider computing a new plan if the current one is actually blocked by a new obstacle (Figure 3.3a, line 6). This keeps the local optimality, since a newly detected obstacle can only imply a costlier plan either moving around it or moving it (**assuming obstacles don't move by themselves, which if it were the case, could open new, better routes**): given that we can assume that the current optimal plan was optimal before the discovery of the new obstacle, we need only reconsider it if the new obstacle invalidates it.

Second Optimization Stop simulating pushes in a given direction before it becomes costlier than the current valid optimal plan, thanks to a bound (Figure 3.3b, line 5). This also keeps optimality, since there are no reasons to continue evaluating actions that are already costlier than simply following the current valid optimal plan.

Third Optimization Only compute the third path component (from obstacle to goal) with A* if the simulated movement actually creates an opening (Figure 3.3b, line 7), since, according to the author, checking an opening creation is less computing time-consuming than running a search algorithm to the goal. The exact opening detection algorithm used is not adressed in this paper and the method used is not explicit. A technical paper later written by co-authors Levihn and Stilman [20], describing their new opening detection algorithm used in [16] however gives a hint at the used method: *"The algorithm did not rely on search but simply observed the amount of adjacent free spaces on corners of the manipulated obstacle. While efficient, this algorithm is only applicable for world configurations populated with simple rectangular shaped static and movable obstacles. This is not realistic."*. This algorithm, like the one presented in the technical paper, only observes the variations of occupied space in the local vicinity of the movable obstacle. In the technical paper, it is explained 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.



(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 translation to the left.

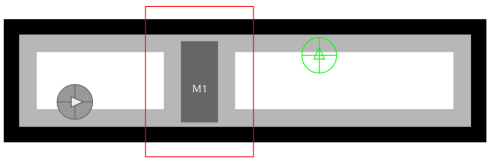
(E) No blocking area has disappeared: no new opening is detected.

FIGURE 3.4: Opening detection example given in the technical paper [20].

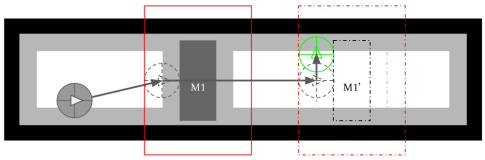
However, with this definition of new opening detection, we also end up with a loss of local optimality, and also lose the algorithm's completeness. Indeed, conditioning the full evaluation of a plan by the detection of a new opening prevents from finding the solution in cases where the local environment of the obstacle does not contain any other obstacle (Figure 3.6), or if for the considered manipulation, the local environment of the obstacle does not change (Figure 3.5). Below are examples figures of these cases, assuming the cost of following a path is the same whether it implies moving an obstacle or not.

In the corridor case initial situation (Figure 3.5a), 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 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, Figure 3.5b 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 (Figure 3.6a), 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 Figure 3.6b that implies moving M1 to its new configuration M1'.

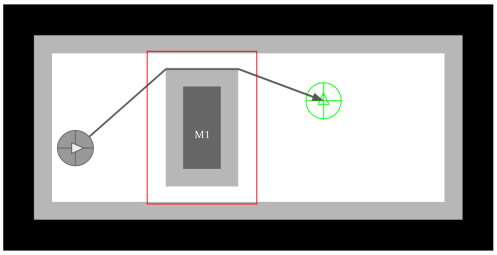


(A) Initial situation where no plan will be found when using new opening checking.

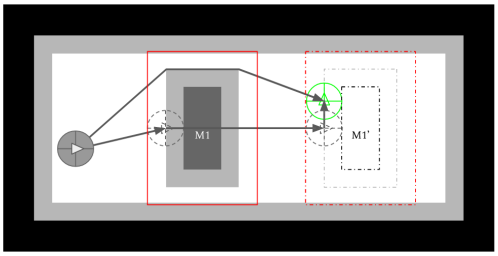


(B) Expected optimal plan.

FIGURE 3.5: "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.

FIGURE 3.6: "Open space" case where the original algorithm will only find a sub-optimal plan.

Below, we propose, but do not provide a full demonstration for a way for restoring the optimality, assuming we are under the hypothesis of sole translations, in a single direction. Since performance is not the main focus of our work here, but optimality is, we will prefer not to use the opening check optimization step in our following algorithms propositions in chapter 4, and postpone a proof to later work.

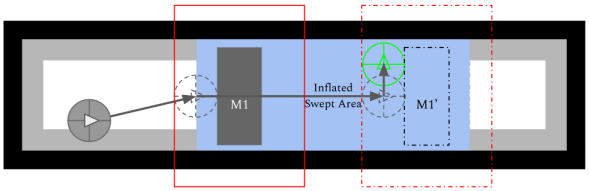
For remembrance, a new opening is never detected if:

- Not a single blocking area disappears thanks to the considered manipulation, because the blocking areas do not vary enough or at all ("corridor" case, Figure 3.7a),
- There are no blocking areas to begin with ("open space" case, Figure 3.7b).

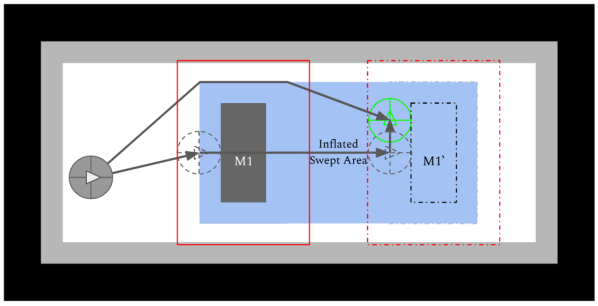
In both cases, it is only interesting to consider the manipulation if it actually creates any chance of finding a plan that has a lower cost than the one that avoids the obstacle.

We have an intuition that, if no new opening is detected, and if, like in the "corridor" case, no plan avoiding the obstacle was found, or, like in the "open space" case, a plan avoiding the obstacle was found, we should only consider the manipulation if it allows us to push the obstacle through the goal pose; in more precise terms, **if the goal pose is within the "inflated swept area" and the obstacle in its final position does not intersect with the goal pose**. The inflated swept area is defined as the area covered by the inflated (by the robot's radius) obstacle when moved. In the end, the overall check condition should be:

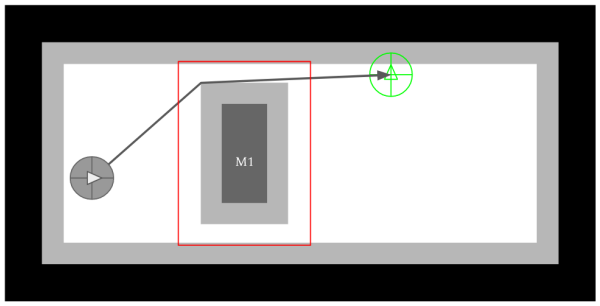
If CHECK-NEW-OPENING($I.occGrid$, o, $translation$, BA) AND $goalPose \in$ GET-INFLATED-SWEPT-AREA(o, $translation$, I) AND $goalPose \notin o.inflatedArea$



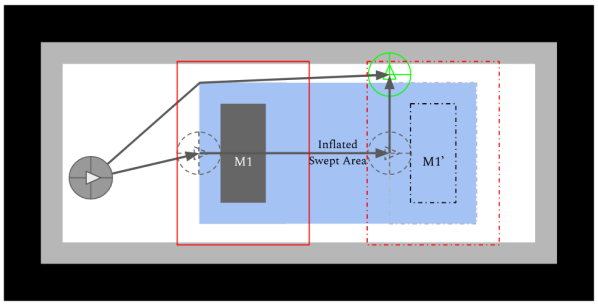
(A) "Corridor" case: with our proposition, the optimal plan will be computed since the goal is in the blue area.



(B) "Open space" case: again the optimal plan will be found for the same reason as in 3.7a.



(C) "Corridor" case with goal outside the "Inflated swept area"



(D) "Open space" case with goal outside the "Inflated swept area": the optimal plan is the one that avoids the obstacle: the plan moving the obstacle will never be computed since it cannot be better than the other one since the goal is not in the blue area.

FIGURE 3.7: Limit cases of the original algorithm with illustration of the "Inflated swept area"

Fourth Optimization Finally, when the plan needs to be re-evaluated, new obstacles are evaluated first(Figure 3.3a, lines 8 to 12) and the resulting paths are saved into a list (Figure 3.3a, lines 2 and 10) by growing order of an underestimated heuristic cost that corresponds to sum of the costs of the plan components c_2 and c_3 (Figure 3.3b, line 12). Then, this list is used to iterate over all obstacle/push direction combinations, and re-evaluate the corresponding plan (Figure 3.3a, lines 13 to 20). The re-evaluation can be stopped as soon as the next pair to consider has a heuristic cost greater than the cost of the current optimal plan, improving execution performance (Figure 3.3b, line 14). However, this optimization step causes the loss of the guarantee of optimality. The heuristic cost depends on c_2 and c_3, and when moving an obstacle, these components' costs may get lower. As the algorithm never updates the heuristic cost (*minCost*) in the list according to this possibility, there is therefore no guarantee that the heuristic cost will always be an underestimate. In the words of Levihn, main author of the second article: *"Second, as the algorithm does not acknowledge the fact that free-space can be created during the execution (e.g. by moving objects), which can lower c2 or c3 for some objects, this optimization steps sacrifices local optimality."* [16]

Note on the use of A* in the main loop In Figure 3.3a, lines 3 and 7, a call to the A* algorithm is made. For line 3, it determines the optimal path from the initial position to the goal, supposing no obstacle has been detected yet. In line 7, after the current optimal path has been invalidated by the detection of a collision between this path and a new obstacle, this call to A* star allows to

get a new optimal path that avoids all obstacles that will serve as basis for the comparison with paths that consider moving obstacles.

3.2 Removing ambiguity

The original pseudocode presented above has quite a few typos, implicit or incongruous notations that create ambiguity (e.g., storing costs and paths in the same variable, having two different variable affectation operators ($\leftarrow$ and $=$), ...), confusing the reading. Since no pseudocode is provided by the authors for the second article's improvements propositions, it was necessary to first fix the pseudocode of the first article. The following paragraphs aim at doing this. The final pseudocode formulation is available in the Appendix section A.1.

3.2.1 Notations and definitions

Paths are ordered sets of "steps", which are themselves robot poses.

A* or D*Lite calls return A PATH as defined above. If no path is found, the returned set is empty: $\emptyset$.

Plans are noted with a lowercase p. Lists or sets of plans will be noted with an uppercase P. A plan is a data structure with a "components" list attribute in which paths are stored in order of execution, and a "cost" attribute that represents the cost of executing the plan.

Components of a plan are noted with a lowercase c.

The norm of a path (written $|path|$), assuming we suppose a cost in distance, corresponds to the sum of the euclidean distances between consecutive steps, and $+\infty$ if the path is empty.

The current robot pose is noted with an uppercase R, the initial pose with R_{init} and the goal pose with R_{goal}

Obstacles are noted with a lowercase o. Lists or sets of obstacles will be noted with an uppercase O.

moveCost **and** *pushCost* are constants without dimension and set to 1 if we consider an optimality in distance. If we wish for optimality in energy or time, we could suppose that they would respectively represent a force or an inverse speed.

one_push_in_d is also a constant representing one elementary push. It is not explicit in the article how it is computed, but we can assume it is the multiplication of a distance constant (likely the resolution of the grid used for A*) by the unit direction vector for the given direction d. We will make this more explicit in our own algorithm

Intersection checks between a plan and obstacles, or between obstacles, are noted with $\bigcap$ and return the set of obstacles that are intersecting, hence the line 13 in Algorithm 1: $p_{opt} \bigcap \mathcal{O}_{new} \neq \emptyset$. In an implementation, this could be done for example by checking whether every pose in the path is not comprised in the inflated obstacles representation, and for obstacle intersection, by checking whether their polygonal representations intersect.

List traversal getNext() is a helper method to traverse a list. It returns null when all elements have been traversed. Variations like getNextStep() work in the same way.

I.withSimulatedObstacleMove is a notation to explicit the fact that the c_3 component must be computed assuming that the obstacle has been pushed, otherwise it does not make any sense.

3.2.2 Ambiguities in the algorithm's logic

Update of the robot's knowledge The first big ambiguity in the algorithm's logic is the GET-NEW-INFORMATION method (Figure 3.2a, line 5). In the original paper, it is not explicit how it works, thus we don't know how it returns a list of new obstacles. We therefore changed it into UPDATE-FROM-NEW-INFORMATION() method (Alg. 1 line 11), that updates the world representation I given in parameter, with new information about obstacles that is collected in parallel in a different execution thread. By that we mean, if this new information includes modifications to known obstacles, they are updated, and if there are new obstacles, they are added to I. We assume that the attribute *I.occGrid* corresponds to the binary occupation grid with inflated obstacles in the current state, so that the path finding routine may run with it. In the same way, I.newObstacles corresponds to the list of newly observed obstacles since the last call of the same method (the affectation to $\mathcal{O}_{new}$ is kept as a shorter placeholder).

Manipulation/Push poses The second bothering ambiguity is that, for a given obstacle, the algorithm iterates over every push direction applicable to it, but doesn't iterate over every point (or manipulation/push poses) from which it could apply said push direction. We must deduce that there is an **implicit hypothesis that for a given push direction, only one point around the obstacle is a valid manipulation start point**. Therefore, we will assume that *o.init* (Alg. 2, line 6) corresponds to the pose the robot must get into in order to move obstacle o in direction d. From the video[2] that presents an implementation of the original algorithm, this pose:

- Is orientated in the given direction, toward the side of the obstacle that allows to push in the given direction,
- Is situated on precomputed "manipulation points" that are at a robot radius distance from the side; often it seems that this point is in front of the side's middle point,
- Among these "manipulation points" it seems the one closest to the current position of the robot is chosen. This is fine since the algorithm seems to operate under the **implicit hypotheses that the friction between the ground and the obstacle is negligible, and that the robot's width is always smaller than the length of the obstacle's side being pushed**. Thus, if the obstacle is movable and not blocked by surrounding obstacles, it will move in the direction it is being pushed in, whatever the accessible "manipulation point" on the appropriate side may be.

As the algorithm only admits pushes in straight lines, c_2 is therefore simply the set containing the initial pose for manipulation *o.init* and the pose where the robot will stop pushing the obstacle *oSimPose*, as in line 15 of Algorithm 2.

Detecting the success of a manipulation Another ambiguity is caused by the lack of a pseudocode element translating the hypothesis given in the article that if the robot tries to move an obstacle but does not succeed, this obstacle will never be considered for manipulation again. It

[2]Experimentation video: https://youtu.be/oQZLbJHYrl8

is meant to allow the robot to detect unmovable obstacles and to avoid an infinite loop caused by an endless evaluation of a same obstacle that cannot be moved. We translate this hypothesis in pseudocode by replacing the vague "$R \leftarrow$ Next step in p_{opt}" statement (Figure 3.2a, line 17) by checking whether the robot actually checking whether robot succeeded or not the desired movement by comparing the actual pose R_{real} after execution and the one we wished to reach R_{next} , and using the *blockedObsL* set to remember obstacles that should never be evaluated again (Alg. 1, lines 40 to 48).

Also, still in the theme of estimating the success, or rather feasibility in this case, of a manipulation, it is not said in the article how the "push on o in d possible" condition (Alg. 2, line 12) is verified. We will assume that it is only checked by verifying that the obstacle's new occupied space doesn't intersect with any other obstacle.

Edge cases and stop condition In order to explicitly manage often occuring edge cases, we have added three conditions that were no originally mentioned. The first, and most important one is the stop condition in case no plan avoiding or moving obstacles has been found (Alg. 1, line 37). If no path has been found even after considering all relevant obstacles, then the algorithm must return a negative boolean value, or the goal is reached a positive one. Two conditions have been added to Algorithm 2, both checking if the path finding subroutine did find a path or not when computing c_1 and c_3 (lines 7 and 17). For c_1, this means there is no valid plan to be found for a given obstacle and push direction. For c_3, this means there is no valid plan to be found for the given pushes in a single direction on an obstacle.

3.3 Pseudocode expression of Levihn's recommendations

In the second article [16], Levihn brings alternate solutions for the Second Optimization, Third Optimization and Fourth Optimization, reducing the computational effort and enlarging the scope of problems the algorithm can manage. For the Fourth Optimization, the changes make it so optimality is no longer affected by this optimization step.

3.3.1 Summary of the paper's modifications on optimizations

Second Optimization In this article, the authors precise that they are using an improved opening detection algorithm, detailed in their separate technical paper [20]. Since, contrary to the previous one, this new algorithm doesn't rely on obstacles being rectangles, but accepts any kind of polygon, it extends the capability of the overall algorithm to any convex polygon. The algorithm is detailed and its formulation is also improved in Appendix section A.8.

However, in the same way we have shown that using opening detection for considering the computation of a full plan affected optimality before (section 3.1), since no measures are proposed in this new article to take this into account, we must assume that local optimality is in fact not restored in this proposition.

Third Optimization The bound that allows to reduce the number of unnecessary evaluations of extra pushes is tightened by adding to the current value of $|c_2|$ (computed as the product between the *pushCost*, the cost of *one_translation_in_d* and the number *seq* of unit translations that have been simulated) the cost of the first plan component c_1 and an underestimate of the cost of the third plan component c_3 (Alg. 5, lines 14 to 16). This underestimate is the euclidean distance between the last position of the simulated push pose *oSimPose* and the goal pose R_{goal}. This bound is thus proved to be an underestimate of the real cost, keeping optimality.

Fourth Optimization Last but not least, a new heuristic is proposed alongside a modified version of the previous one. Basically, all obstacles that haven't been evaluated at least once are ordered in a separate list *euCostL* by a heuristic cost that is independent from c_2 and c_3: the euclidean distance between the goal pose R_{goal} and the obstacle's nearest "manipulation point" at which the robot could manipulate it. When an obstacle has been evaluated, it is added to another list *minCostL* ordered by the usual *minCost*. Since this heuristic is more informed, *minCostL* is used first when available (Alg. 4 line 9). If not, *euCostL* is used, the obstacle is re-evaluated and naturally added to the list ordered by *minCost* (Lines 22 to 34). This is achieved through the use of separate indexes for traversing the lists: i_e and i_m. If the next entry from *minCostL* or *euCostL* to be considered (the one with the lowest cost) is associated with a cost that is greater than the current optimal plan, then it is not worth trying to evaluate any more options, and therefore the loop must end (Alg. 4 line 8). The heuristic list *minCostL* depending on the costs of c_2 and c_3 is invalidated (emptied) anytime an obstacle has changed of place (which can potentially lower the cost of c_2 or c_3). Thus, local optimality is no longer affected. For a more detailed explanation, please consult the pseudocode (Appendix section A.2), or the original article [16].

N.B on pseudocode restructuration Since the pseudocode is our own interpretation, for easier understanding, we took the liberty of reorganizing the algorithm structure given in the original proposition of Wu [38]. We separated the plan execution and validity verification code (corresponding to lines 2 to 16 and 35 to 50 in Alg. 1) from the code that manages the order of evaluation of obstacles (corresponding to lines 17 to 34 in Alg. 1) into two separate methods that reflect this (Algorithm 3: "MAKE-AND-EXECUTE-PLAN"), and (i.e Algorithm 4: "MAKE-PLAN"). We also renamed the "OPT-EVALUATE-ACTION" method (Algorithm 2) into "PLAN-FOR-OBSTACLE" (Algorithm 5) to show that our method evaluates all possibilites for a given obstacle rather than an obstacle AND a direction.

3.3.2 Newly introduced notations and definitions

[] operator For the sake of readability in the pseudocode, if the list element that is asked for is out of bounds (empty list or reached end of list), the "[]" operator shall return a "fake" tuple with a null obstacle reference, and infinite cost: {null, $+\infty$}. This could easily be implented in code by either using a ternary operator (for example, "$minCostL[i_m].minCost$" would become "$minCostL[i_m] = \text{null} \ ? \ +\infty \ : \ minCostL[i_m].minCost$") or implementing a custom array object with the wanted behaviour.

evaluatedObstacles is a set that remembers which obstacles have been evaluated in the current MAKE-PLAN instance (Alg. 4, lines 7, 11, 18, 24 and 31). This is not mentioned in the original article, but it avoids evaluating an obstacle twice when it is added to *minCostL*.

I.**freeSpaceCreated()** This method is used in Alg. 3, line 12, to allow the invalidation of *minCostL* as previously explained in the Fourth Optimization paragraph. It returns True if any obstacle's occupied space has been reduced, False otherwise.

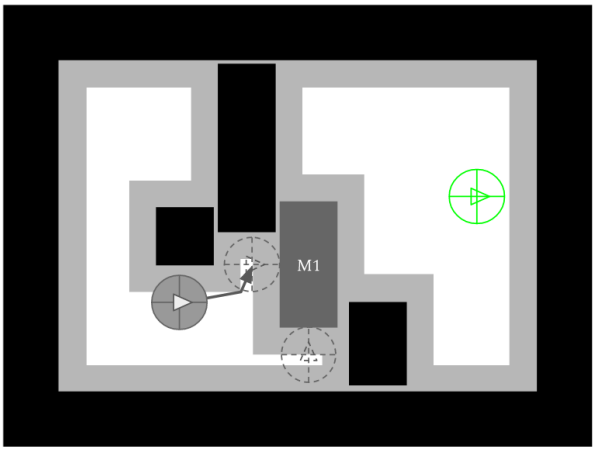
I.allObstacles is the list of all observed obstacles in the current state (Alg. 3, line 16), in the same fashion as *I.newObstacles*.

BA is a buffer for the initial blocking areas when using the new algorithm for more efficient opening detection (Alg. 5, lines 10 and 17). On the first call, the variable is initialized with the initial blocking areas, and at each following call, it is passed as a parameter to reduce computational overhead. This measure is recommended by the technical paper.

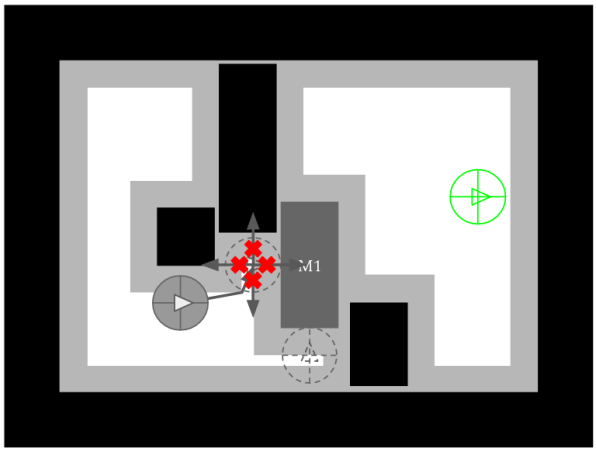
3.3.3 New ambiguities in the algorithm's logic

The first ambiguity in the second article [16] is that the D*Lite algorithm is mentioned instead of A*, but not even a hint of an explanation is given as to why this change, or what difference in the implementation it makes. The main feature of D*Lite being that it is designed for navigation in dynamic environments, given that no propositions are given to adapt the rest of the algorithm to dynamic environments, we will assume that no particular advantage is obtained by using D*Lite rather than A* in the proposition. Therefore, in our own proposition presented in the next chapter, we shall continue using A* as our path finding subroutine.

The second ambiguity is about manipulation poses, again, as in previous section. In the article, the authors claim that *"c1 only needs to be calculated once for the entire process of evaluating the current object."*. With the same reasoning as in the previous paragraph Manipulation/Push poses, this affirmation can only be true if the algorithm operates under the **implicit hypothesis that for all given manipulation directions, only one point around the obstacle is considered a valid manipulation start point**. From the video accompanying the article, this point seems to be the nearest point from the robot, situated at a radius distance from the middle of a side of the obstacle. However, this hypothesis actually hinders optimality: if there is in fact a valid manipulation point for each side of the obstacle, and the algorithm knowingly doesn't consider them because they are further from the current robot's position, it will ignore the fact that a same manipulation direction could end up in opening a better path if the obstacle were moved from another manipulation point (see figures below). To guarantee optimality, we would have to simulate the manipulation in the given direction for every reachable manipulation point, thus re-evaluating c_1 for each. That would result in adding an extra "for" loop englobing the existing one. This is illustrated by the figures below.



(A) Nearest manipulation pose doesn't allow moving the obstacle at all.



(B) But if we also consider the other pose, a valid plan can be found.

FIGURE 3.8: Illustration of the importance of considering all possibles manipulations for all manipulation poses and not just considering the nearest one.

With all the ambiguities out of the way, we now have a solid foundation on which to build upon our own proposition, according to our own hypotheses: this will be the focus of the next chapter.

Chapter 4

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

4.1 Discussion on the original hypotheses in the light of tests with the Pepper robot

The pseudocode formulation A.3 for this proposition is available in Appendix A.

As eventually our experimental platform is to be a Pepper robot in the context of the Robocup@Home challenge that emulates a home setting, several hypotheses from the original algorithms 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. 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, since they use this environment knowledge to compensate for odometry error.

- **Manipulation actions**, for the moment, are to be limited to pushes in a perpendicular direction to the obstacle's side being pushed. 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.

- **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 Chapter 5) and cardboard 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. 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).

- **Manipulation cost** A constant *pushCost* has been used in the previously shown algorithm to allow weighting of the manipulation action in regard to a simple move action. Semantically, it makes more sense that this constant be related to the object (the difficulty of moving a specific object depending mainly on its physical properties), so we will store it as an obstacle attribute.

- **Manipulation possibility check** Checking whether a manipulation is possible or not is done by checking whether the area covered by the robot and the obstacle as they move together is in intersection with any other obstacle. As we limit our action set to pushes in a specific 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. According to the existing litterature, we will call this the "safe-swept area" if no other obstacle is in intersection with it. In the pseudocode, this is done by the "GET-SAFE-SWEPT-AREA" method, which returns null if any obstacle is in intersection with the manipulation area. This area is saved as part of the plan so that when the plan is being executed, checking for a collision is as simple as checking if an obstacle appeared in this area (assuming our knowledge of the obstacle did not evolve in the mean time).
- **Obstacle discovery** 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, since the obstacle's push poses may change (as a push pose has been defined with a dependency to the side's middle point). 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.

4.1.1 Newly introduced notations and definitions

continue The **continue** statement returns the control to the beginning of the loop, and simply won't execute any of the remaining statements in the current iteration of the loop. This is done because a plan with a manipulation cannot exist without an empty c_1 component (i.e. the targeted push pose is not accessible).

COPY Here, $p_{opt}.o$ is a copy of object o, and not the same object, so that when o is updated because of the call to UPDATE-FROM-NEW-INFORMATION() on I, we can compare the difference between the two. We do the same for $p_{opt}.pushPose$ for the same reason. This allows us to trigger re-evaluation if the obstacle's push poses change and the one the robot aimed for no longer exists.

[], $\neq$ and $\bigcap$ operators The [] operator is now also used as a short handle for "get the obstacle that corresponds in $\mathcal{O}$ that corresponds to the saved obstacle $p_{opt}.o$. The $\neq$ operator checks if the two states of the obstacle are the same or not (i.e, if the obstacle changed).

$\bigcap$ The notation $\bigcap$ means here that we check for possible collisions between the swept area and any obstacle, since they may have changed.

Note on saving *translation* in p_{opt} The *translation* necessary for manipulating the obstacle is saved to easily recompute the safe swept area when the obstacle changes.

4.2 Social awareness through manipulation authorization consideration

As shown in Chapter 2, to the best of our knowledge, the current NAMO litterature has never covered the idea of socially-aware navigation. Then, we must ask: what makes the action of moving an obstacle socially-aware or not?

The first thing that comes to mind would be to consider that some objects are better not be moved because:

- they are too fragile (e.g. flower pot),
- they have a high value in the humans eye (e.g. a costly vase)
- they might cause the robot to break if it fails to move them properly (i.e. heavy or unstable objects)
- they are not supposed to be moved (i.e. exhibited objects)
- ...

Thus comes the notion of risk, either to the robot or to the manipulated objects. To mitigate this risk, we propose to modify our base algorithm presented in the first section of this chapter, so that an obstacle is not to be moved unless identified as belonging to a provided whitelist of "movable" obstacles.

We will work under th hypothesis that if an obstacle is in the field of vision of the camera, the obstacle recognition will be perfect if it is in our whitelist of movable ostacles. If an obstacle is in the field of vision of the camera but is not recognized, we will assume that it is an obstacle that is not supposed to be moved. This is relatively safe hypothesis if identification of the obstacle's nature is done through computer vision, since it is one of the most efficient and most common ways to detect specific objects, by using trained neural networks for example[25]. However, robots come with all sort of sensors to detect obstacles: laser range finders, RGB(D) cameras, sonars, ... And often, as with Pepper (see Chapter 5 for a description of Pepper's abilities), their fields of vision do not perfectly overlap: typically, an obstacle may be detected by the laser range finders or the sonars, but not be within the field of vision of the RGB(D) camera, because it is in its blind spot or simply too close or too far away. This creates a situation where the robot knows an obstacle is there, but cannot definitely categorize it as "movable" or "unmovable" since it is not in the camera's field of vision.

Then, it means that the algorithm must be adapted not only to manage the fact that an object should be considered for manipulation if and only if it is not deemed "unmovable", but also to eventually adapt the robot's trajectory **in an optimal way** so that an "unidentified" / "potentially movable" object can be identified with certainty before engaging with the manipulation procedure.

For that, when we evaluate an obstacle, we first check whether the obstacle has already been identified or not (Alg. 9, lines 2 and 13). If it has been identified as "movable", the algorithm does not change. If it has been identified as "unmovable", the obstacle evaluation routine simply stops before actually evaluating. And finally, if the evaluated obstacle is "unidentified":

- The c_1 plan component that goes from the current robot pose to the push pose is evaluated just as before (Alg. 9, line 8),
- If a pose comprised in the computed c_1 component allows the camera field of vision to encompass the obstacle's currently known geometry, keep the precomputed c_1 component (Alg. 9, lines 14 to 16, and Alg. 10),
- Else we must find a shortest path component c_0 from the current robot pose to an "observation pose" where we know we can identify the obstacle as "movable" or "unmovable" with certainty and recompute c_1 as the path from this "observation pose" to the push pose. Recomputing is done in the COMPUTE-C0-C1 method (Alg. 11), which is a baseline, naive implementation where we iterate over every "observation pose" to compute a path for c_0 and c_1 and only keep the shortest total path (this is illustrated in Figure 4.2). If neither c_0 or c_1 components are found, then stop evaluating plans for the current push pose (Alg. 9, lines 19 to 21). Since the robot's obstacle representation may change as the robot approaches it, the condition favors paths combinations with an observation point that is closest to the current robot's pose, so that there are more chances for the robot to still have a valid observation pose in the current plan avoiding the need to recompute a plan in some cases.
- The observation poses are updated in the same way that push poses are: automatically, whenever an obstacle is updated. These poses are situated at every grid point for which the field of vision of the robot sensor(s) dedicated to obstacle recognition covers the entire known obstacle's geometry. Though the presented algorithm is not affected by the representation of the identification sensor's field of vision, in our experimentation, we will consider a single RGB(D) camera, and approximate its field of vision by the difference between a circular sector and a disk of same center, coincident with the robot's center, which is an acceptable representation for the Pepper robot capabilities. The circular sector has a radius r_{max}, central angle θ and is equally partitioned around the robot's orientation direction line. The disk has a radius r_{min}. This is illustrated in Figure 4.1.

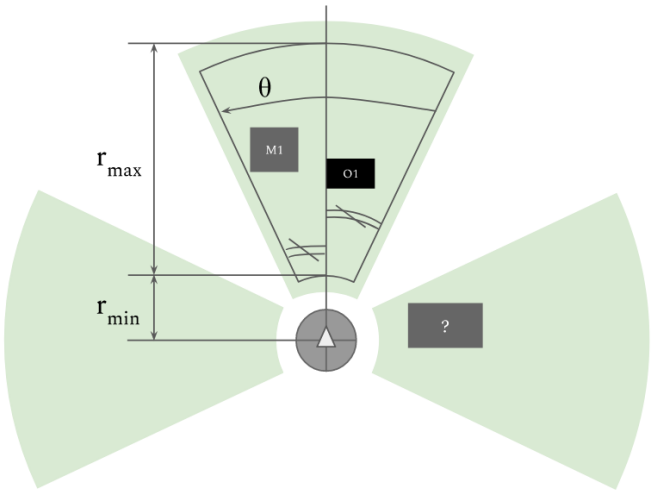
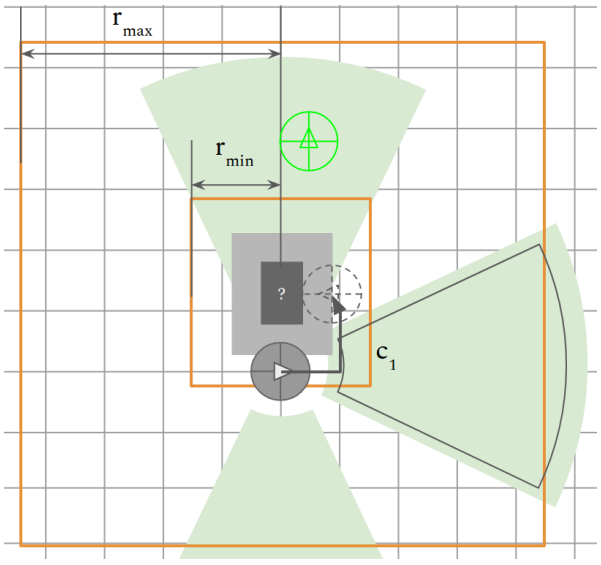
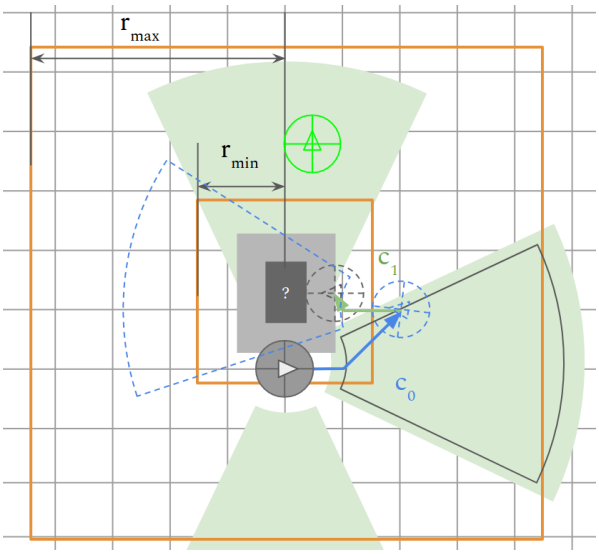


FIGURE 4.1: Illustration of the field of vision of the different sensors of the Pepper robot: the green circular sectors correspond to laser sensors that only provide information about the obstacle's geometry. The grey line sector is the FOV of the camera used for identification. We can see that movable obstacle M1 and unmovable obstacle O1 have been identified but another obstacle "?" is yet to be.

In 4.2a, the robot is in a situation where its laser sensors (green) have detected an obstacle to its left, but the obstacle has not yet been identified. When evaluating possible manipulation plans, the robot will need to build a c_1 component to every push pose of the obstacle (like the one in grey dots on the right). However, it is clear that here, the robot is too close, and the c_1 component (black arrow) being the shortest path to the push pose, it will not allow the camera FOV to encompass the obstacle, thus preventing identification. In 4.2b, we can see an example application of our proposition that would allow the robot to find an optimal plan in two components c_0 (in blue) and c_1 (in green) that allows it to identify the obstacle.



(A) Initial situation: goal is the green disk with a triangle, and the zone where the robot could potentially see the obstacle is delimited by the area between the two orange rectangles. The observation poses are situated at every grid intersection within the area and are orientated toward the robot's center.



(B) The intermediary pose that is the goal of c_0 is the blue disk, and we can see that the camera FOV (blue dotted circular sector) would then encompasses the obstacle, allowing for an identification decision.

FIGURE 4.2: Example situation showing the use of the c_0 component.

In order to keep our local optimality property, we also modified the main execution loop, see Algorithm 8 (changes are highlighted in green). Basically, we added a check after the robot gets the next step to be executed and its parent plan component: if the currently considered optimal plan implies moving an obstacle that hasn't been identified yet, the next step component is c_0 or c_1, and the obstacle has changed since the previous environment observation in a way that the current plan does not allow to identify it anymore, then we must not execute the next step but re-evalutate the plan, since it may no longer be optimal.

Optimized version It is obvious that the COMPUTE-C0-C1 method can be optimized in terms of execution time, in an analogous way the original algorithm did, by reducing calls to A*. A heuristic cost defined as the sum of the euclidian distance between the current pose and observation pose, and euclidean distance between observation pose and currently evaluated push pose is computed for every observation pose (Alg. 12, lines 4 to 4), and allows to order them in a list *euPosesCostL*, sorted by ascending heuristic cost. The list is then traversed until the heuristic cost of the current element is greater than the current optimal cost of c_0 + c_1 likely allowing not to evaluate many observation points (Alg. 12, lines 8 to 21).

Future work It would be interseting to do a performance comparison between the two versions of our COMPUTE-C0-C1 method, and maybe further optimizing it by using a fitter algorithmic approach than calling A* many times. For example, using Dijkstra's algorithm to first get the path to all observation points in a single graph search rather than many could lead to better performance, especially if there is a great number of observation poses to check.

4.3 Social awareness through placement consideration

Now that we have a basic capability for the algorithm to deal with obstacles depending on whether they have been explicitly defined as "movable" or not, we have an answer to the question "what obstacles can be moved in a socially-aware manner?". But then, this only characterizes the obstacle itself, and not the action of moving it. When moving an obstacle, not only do we have to consider the obstacle we are moving, but also where we are moving it: it would definitely not be acceptable for a robot to move an obstacle in an area where it would impair other actor's movements (e.g., by blocking an entryway, a corridor, ...).

Handling this, while keeping the local optimality property, can be achieved by redefining the cost of a plan not just as being dependent on the distance, time or energy, but also on the compliance to the social rule of not placing obstacles in specific places that we will call "social cost".

In this proposition, we define "socially forbidden/allowed" areas as cells of a 2D grid costmap, that have a value comprised between an arbitrary minimal integer value ALLOWED_VALUE and maximal value FORBIDDEN_VALUE, in an analogous way to the costmap_2d of ROS[1]. Since an obstacle may occupy several points, the total cost of moving an obstacle to some place is expressed as the normalized sum of the costs of covering each point (See Alg. 14, line 11). If a cell is associated with the minimal value then it means that there is no particular wish to avoid covering it with an obstacle, not affecting the total cost (Alg. 14, lines 8 to 10). On the contrary, if it is associated with the maximal value, no obstacle should ever be placed over it, raising the total cost to $+\infty$ (Alg. 14, lines 6 and 7). This total cost is then simply added in the computation of the cost of the c_2 plan component as a product (Alg. 13, highlighted lines).

Future work Though here, we limit ourselves to a static costmap of "socially forbidden" areas, nothing actually stands in the way of updating it according to the data the robot collects about the environment. This could allow to detect inappropriate areas for leaving obstacles that depend on other moving or movable obstacles (e.g., behind a chair, around a wheeled table, ...). This could also, if the algorithm were also modified to handle autonomously moving objects, allow to dynamically attribute a higher placement cost in areas where other agents like humans are about to pass through (for example, by building upon existing work, as in [10], where a method for mapping the likelihood of encoutering humans in an environment is described): we would not want the robot to put an obstacle right in front of a human or a robot minding their own businesses.

4.4 Taking dynamic obstacles into account

A big missing piece in the existing NAMO algorithms is that they often, as is the case with the original one we are building upon here, operate under the assumption that there are no other autonomous agents around. Therefore, an obstacle cannot move of its own volition. For the home environment setting we are aiming for, this is only acceptable if the inhabitants are not

[1]ROS documentation page: http://wiki.ros.org/costmap_2d

here during the robot's activities, and neither pets or other robots are. However, one of the main points of having a service robot at home is to actually interact with it. Therefore, we must at least adapt the algorithm not to enter in collision with a moving obstacle (which it would in its current state since it only invalidates a plan when **new** obstacles are found to be intersected with), and keep making locally optimal decisions.

For that, it is only necessary to modify the main execution loop, since the state of the robot's knowledge about the environment only ever changes here: see highlighted lines in Algorithm 15. First thing to do is to now consider all obstacles when checking for an intersection with the current plan: $\mathcal{O}_{new}$ is no longer useful, thus removed.

To keep local optimality, we chose the simplest approach: whenever an obstacle is detected as having moved, we trigger a plan re-evaluation, though we don't invalidate the current one if it is still valid (according to the same criteria as in Algorithm 6). For that, we assume that, when updated, the world state representation I saves in a list $I.movedObstacles$ the obstacles that have moved since its last update by checking whether they don't occupy some of the space they previously occupied. If this list is not empty, then we must trigger a re-evaluation. Also, since in terms of computation time, the plan evaluation is by far the longest element, if we just went through it, then we do not directly execute the plan but go back to the beginning of the loop and check the environment again. If no obstacles moved since then, the plan will be executed. Otherwise, it will be recomputed. This way, local optimality is always guaranteed.

Future work While the current proposition retains local optimality, if obstacles are constantly moving around the robot, then it will never budge, always recomputing plans, until its the environment in its field of vision stops moving: this is known as the "robot freezing problem" [35]. Furthermore, autonomously moving obstacles follow trajectories that can usually be predicted, at least in a short time window: it may not be interesting to invalidate the current plan and/or trigger a plan re-evaluation if an obstacle just passes by the robot in a way that would not affect the optimality of its plan. Incorporating existing work on taking obstacle trajectory predictions into account will be the object of future study. It may also be interesting to study how to merge the existing NAMO algorithms with an existing algorithm for navigation among dynamic obstacles like D* or D*Lite in a way that actually takes advantage of the incremental building of knowledge of these algorithms.

4.5 Algorithm proposition

The final algorithm proposition is a merge of the previously presented individual propositions. It consists of Algorithms 16, 4 and 17 and the newly created subroutines 12, refalg:04-custom-observation-simple-checkpath and 13. All these can be found in the Appendix A.

Chapter 5

Experimentations & Validation

5.1 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.

5.1.1 Pepper robot characteristics

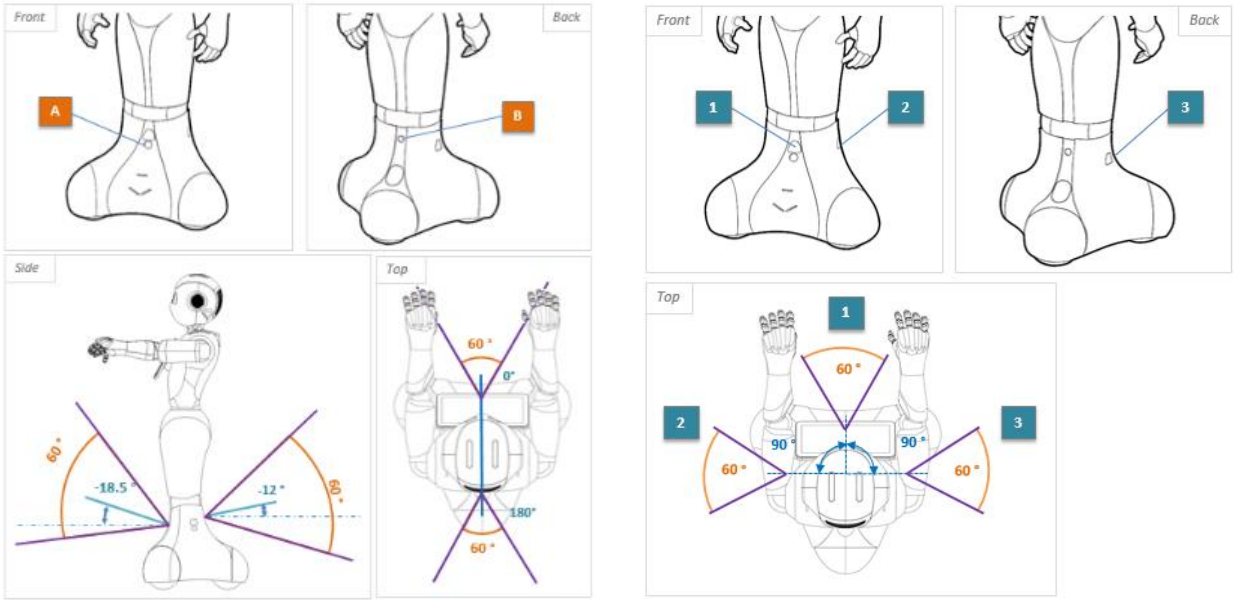
The Pepper robot is a mass-produced humanoid service robot by the company Softbank Robotics [1]. For sensors, it is equipped with:

- Two sonars, at the front and the back (Figure 5.1a), that allow to detect obstacles in the close vicinity, but not to characterize their geometry (with them, the robot can only know that an obstacle is in front or behind, at a certain distance, but cannot discern its form),
- Three laser sensing units at the front and the sides (Figure 5.1b), that allow it to roughly detect the form of nearby obstacles (only 15 points per sensor at a maximum distance of 3 meters). It is to be noted that, together, these three units do not cover the surroundings of the robot at 360 degrees, but leave very big blind spots,
- Three bumpers at each corner of its triangular base, to detect collisions that may occur,
- A 2D RGB camera on its forehead (Figure 5.2a), that allows it to do image recognition tasks or simply filming its environment,
- A 3D RGBD camera [2] in its head (Figure 5.2b), behind the eye lenses, that allow it to construct a 3D point cloud of the obstacles in front of where its face is turned, which can be used to cover part of laser units blind spots,
- An inertial measurement unit that allows it to know its orientation.

In particular, three other sensors are especially meant for interaction with humans:

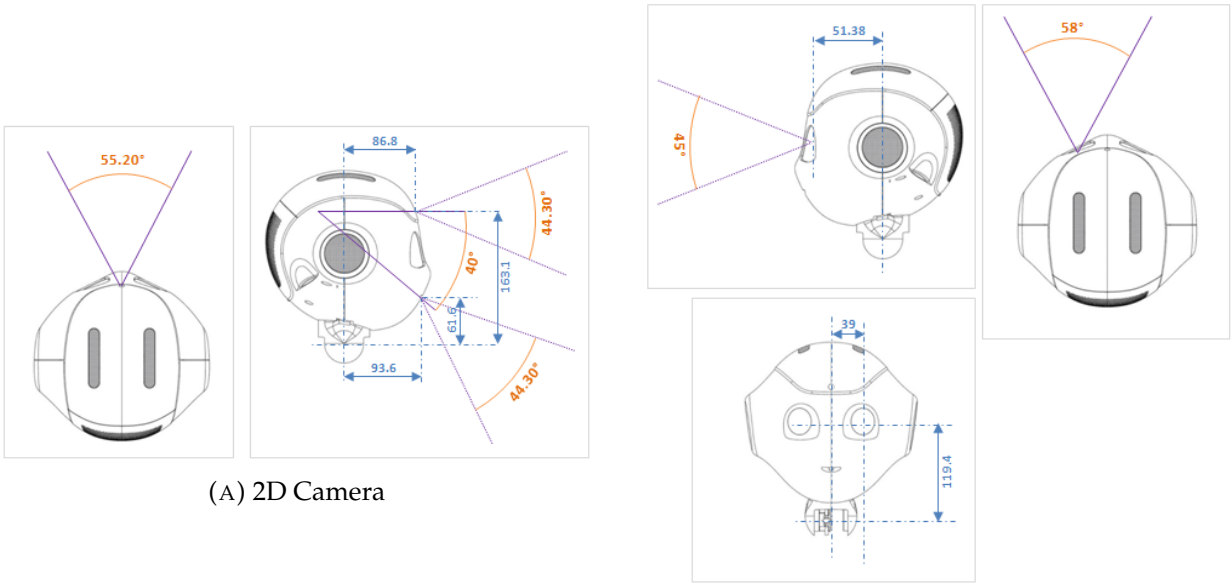
- A touchscreen (also an actuator) on its chest, for complex interactions,
- Two directional microphones on the top of its head that can be used to detect where a noise is coming from or interact vocally,
- Various touch sensors over its body, including on the hands, that allow it to interact physically.

[1] Softbank Robotics website: https://www.softbankrobotics.com/emea/en
[2] Asus XTion Camera: https://www.asus.com/us/3D-Sensor/Xtion_PRO_LIVE/



(A) Sonars (B) Lasers

FIGURE 5.1: Positioning and field of vision of Pepper's sonars and lasers



(A) 2D Camera

(B) 3D Camera

FIGURE 5.2: Positioning and field of vision of Pepper's cameras

As for actuators, Pepper is equipped of omnidirectional wheels that allow it to turn in place and not be forced to face the direction it is translating toward. Two motors allow it to fold its body a little, just above the base and at the hips, but not enough for it to be able to touch the ground with its arms. Two motors allow the head to move forward/backward and turn left or right (but not for a full turn). Finally, it has two articulated arms including fingers, that are mainly meant for interactions with humans, but not for lifting large or heavy obstacles.

All these characteristics are important, because they condition the way we can implement a navigation algorithm.

5.1.2 On experimentation repeatability with ROS and Pepper

We mentioned at the beginning of our work that the INRIA Chroma team uses ROS [3] to program Pepper. ROS is a robotics middleware, and, in fact, not an operating system. It provides a high-level abstraction of hardware while still allowing for low-level device control, task distribution over a network capabilities, and most importantly, a message-passing system between processes, that allows a great diversity of programs written in many different languages to still be able to talk to one another. It also brings a package management system, that is coupled with the packaging system of the GNU/Linux distribution Ubuntu [4], allowing for plug-and-play capabilities for new software components.

Experimentation reproducibility has been a long standing problem in robotics [9, 4]. As an example, in the entire studied corpus presented in Chapter 2, no paper links to actual code or a Virtual Machine (VM) to test the simulations. Even using a standard like ROS does not make an experiment done with it reproducible in itself, since one needs the exact same version of ROS, the same GNU/Linux distribution and version, the same libraries, the same robotic platform with the same peripherals (when using a real robot), ... To solve this problem at least on a software level, in the last few years, the ROS foundation financed efforts toward using ROS inside of Docker [5] containers [36], a technology close to the idea of a VM, but lighter and faster, since it does not isolate the running code as much as a VM.

We have therefore built upon this work to propose a Docker container configuration that allows to connect to the Pepper Robot and reproduce the following experiment on any Linux-Enabled system. All instructions required to set it up are available in the following git repository: `https://github.com/Xia0ben/rosdocked-kinetic-pepper`.

5.1.3 Experiment

Objective We want to test several house-environment movable obstacles types to see if and how Pepper could manipulate them with confidence as to the results.

Hypothesis Given the round nature of Pepper's base front, we suppose that light obstacles and obstacles with wheelcasters could be moved. We suppose 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.

Protocol

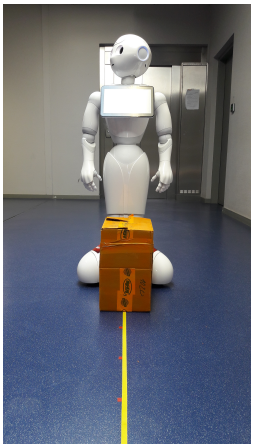
- We start a Docker container to connect to the robot, following the instructions in the previously presented repository.
- Position Pepper at the beginning of the test line with an obstacle at its base in a given configuration (for each obstacle, we at least test the middle of the obstacle's side, and 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), see Figure 5.3.
- 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

[3] ROS, the Robot Operating System, website: `http://www.ros.org/`
[4] Official website: https://www.ubuntu.com/
[5] https://www.docker.com/

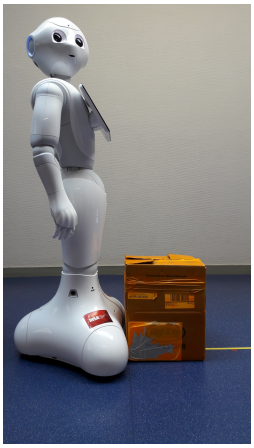
```
\$ rostopic pub /cmd_vel -1 geometry_msgs/Twist\
'{linear:  {x: 0.5, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'\
&& sleep 1 &&\
rostopic pub -1 /cmd_vel geometry_msgs/Twist\
'{linear:  {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'
```

- 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.

(A) Front close angle

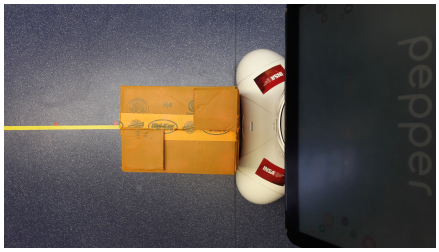


(B) Front wide angle



(C) Side close angle

(D) Side wide angle



(E) Top close angle

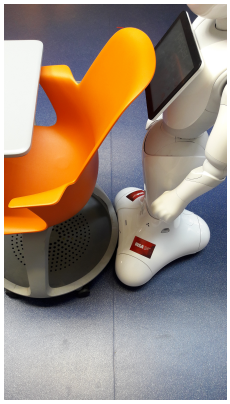
FIGURE 5.3: Experimental setup example with a small cardboard box

Results

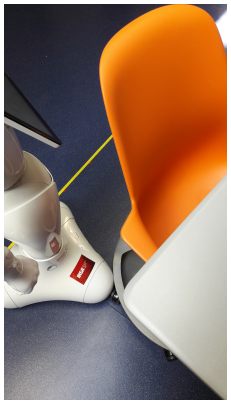
After tests conducted on empty cardboard boxes of varied sizes (bigger or smaller than the robot's base), a chair with wheelcasters and an office trash can, we experimentally conclude that:

- 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 (Figures 5.4a and 5.4b).
- 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 (Figures 5.4c and 5.4d), 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).

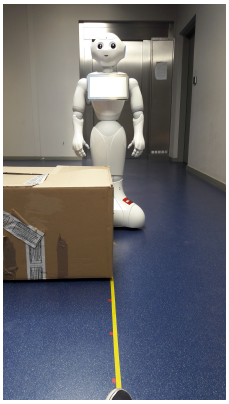
Conclusion Therefore, it is relatively safe to assume that always placing our robot at the middle of the obstacle's side for a push action will almost always produce the expected translation without significant drift.



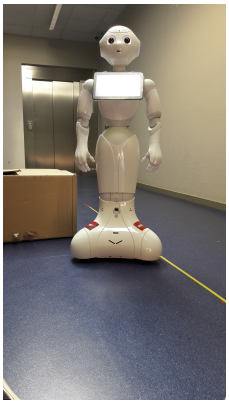
(A) Initial top close angle of well-positioned robot and chair



(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

FIGURE 5.4: Failed manipulation attempts with badly positioned robot and a carboard box, and with well-positioned robot and a chair with wheelcasters.

5.2 Simulation in a ROS-Standards compatible simulator

5.2.1 Simulator presentation

In order to test our propositions in Chapter 4, and validate our many improvements over the original algorithm formulation in Chapter 3, we developed a custom simulator based on ROS standards. By that, we mean that the simulator uses ROS standard message types[6], 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 by RViz[7] (Figure 5.5), the standard 3D visualizer for ROS, simply by listening to the 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 *move_cost* and *push_cost* currently present in the algorithm and passing them to the A* path finding subroutine.

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

[6] See ROS documentation page: `http://wiki.ros.org/common_msgs`
[7] See ROS documentation page: http://wiki.ros.org/rviz

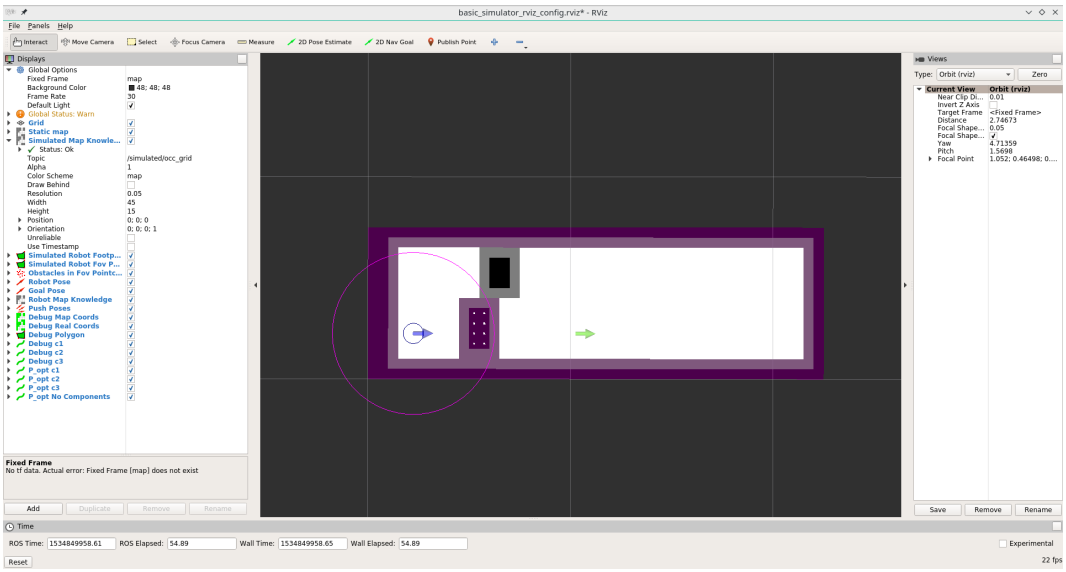


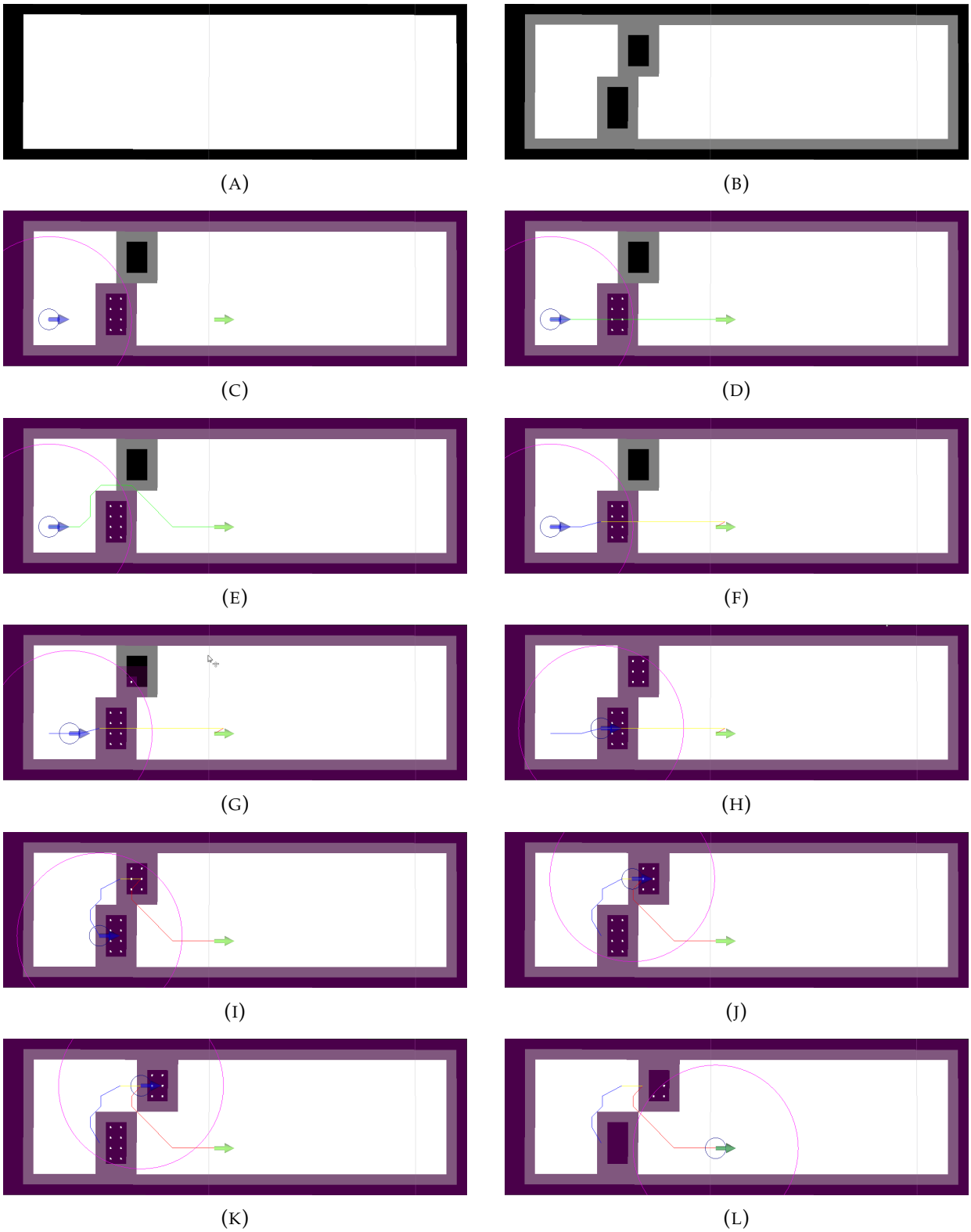
FIGURE 5.5: Rviz interface screenshot. On the left, we can see the visual components used and the message topics they listen to. In the center, we can see in real time the execution of the algorithm.

5.2.2 Simulation results

At the time of the writing of this report, we were only able to implement the algorithm and the necessary simulation code for the solution presented in section 4.1 (and corresponding algorithms 6, 4 and 7), 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. Below are a few figures (5.6) showing a test case in a corridor with two obstacles.

Figure 5.6a represents the static obstacles that **never change** (walls). Figure 5.6b also represents the yet unknown obstacles (the bottom one is unmovable, the top one is movable). Figure 5.6c 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 Figure 5.6d, 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 Figure 5.6e, 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 Figure 5.6f, 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 Figure 5.6g, where we see that is starts detecting the shape of the top obstacle, only to realize in 5.6h that the obstacle won't budge. Therefore, it invalidates the current plan, and computes a new one (seen in Figure 5.6i) that implies moving the top obstacle directly, since no plan avoiding the two obstacles could be found. In Figures 5.6j and 5.6k, the robot moves the top obstacle, to finally reach its goal in Figure 5.6l.



(A) (B)

(C) (D)

(E) (F)

(G) (H)

(I) (J)

(K) (L)

FIGURE 5.6: Simple simulation in a corridor with a movable (top one) and an unmovable (bottom one) obstacles with our algorithm as presented in sectoin 4.1.

Chapter 6

Conclusion and Perspectives

In this work, 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 analyzed the state of the art algorithms for locally optimal NAMO in unknown environments [38, 16], 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.

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 [1]). 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 [15]. 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 [34, 18, 15, 30]. Finally, evaluating this work in the actual context of the next Robocup@Home challenges would be a great way to validate it.

Appendix A

Algorithms

A.1 Reworked Wu's algorithm (as desc. in 3.2)

Algorithm 1 Optimized algorithm for NAMO in unknown environments of Wu et. al. (2010), fixed - MAIN LOOP

procedure OPTIMIZED(R_{init}, R_{goal})
  $R \leftarrow R_{init}$
  $blockedObsL \leftarrow \emptyset$
  $P_{sort} \leftarrow \emptyset$
  $\mathcal{O}_{new} \leftarrow \emptyset$
  $isManipSuccess \leftarrow True$
  $I \leftarrow empty\_occupation\_grid$
  $p_{opt}.components \leftarrow [A^*(R_{init}, R_{goal}, I.occGrid)]$
  $p_{opt}.cost \leftarrow |p_{opt}.components[0]| * moveCost$
  while $R \neq R_{goal}$ **do**
    UPDATE-FROM-NEW-INFORMATION(I)
    $\mathcal{O}_{new} \leftarrow \mathcal{O}_{new} \bigcup I.newObstacles$
    $isPathFree \leftarrow p_{opt} \bigcap \mathcal{O}_{new} \neq \emptyset$
    if not($isPathFree$ AND $isManipSuccess$) **then**
      $p_{opt}.components \leftarrow [A^*(R, R_{goal}, I.occGrid)]$
      $p_{opt}.cost \leftarrow |p_{opt}.components[0]| * moveCost$
      for each $o \in \mathcal{O}_{new}$ **do**
        for each possible push direction d on o **do**
          $p \leftarrow$ OPT-EVALUATE-ACTION(o, d, p_{opt}, I, R, R_{goal}, $blockedObsL$)
          if $p \neq$ null **then**
            P_{sort}.insert(p)
          end if
        end for
      end for
      if $P_{sort} \neq \emptyset$ **then**
        $p_{next} \leftarrow P_{sort}[0]$
        while $p_{opt}.cost \geq p_{next}.minCost$ AND $p_{next} \neq$ null **do**
          $p \leftarrow$ OPT-EVALUATE-ACTION($p_{next}.o$, $p_{next}.d$, p_{opt}, I, R, R_{goal}, $blockedObsL$)
          if $p \neq$ null AND $p.cost \leq p_{opt}.cost$ **then**
            $p_{opt} \leftarrow p$
          end if
          $p_{next} \leftarrow P_{sort}$.getNext()
        end while
      end if

$\mathcal{O}_{new} \leftarrow \emptyset$
end if
if $p_{opt}.components = \emptyset$ **then**
return *False*
end if
$isManipSuccess \leftarrow True$
$R_{next} \leftarrow p_{opt}$.getNextStep()
$c_{next} \leftarrow p_{opt}$.getNextStepComponent()
$R_{real} \leftarrow$ ROBOT-GOTO(R_{next})
if $c_{next} = c_2$ 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 2 Optimized algorithm for NAMO in unknown environments of Wu et. al. (2010), fixed - ACTION EVALUATION SUBROUTINE

procedure OPT-EVALUATE-ACTION(o, d, p_{opt}, I, R, R_{goal}, $blockedObsL$)
if $o \in blockedObsL$ **then**
return null
end if
$P_{o,d} \leftarrow \emptyset$
$c_1 \leftarrow A^*(R, o.init, I.occGrid)$
if $c_1 = \emptyset$ **then**
return null
end if
$c_2 \leftarrow \emptyset$
$oSimPose \leftarrow o.init$
while push on o in d possible AND $|c_2| * pushCost \leq p_{opt}.cost$ **do**
$oSimPose \leftarrow oSimPose + one\_push\_in\_d$
if push created new opening **then**
$c_2 \leftarrow \{o.init, oSimPose\}$
$c_3 \leftarrow A^*(oSimPose, R_{goal}, I.withSimulatedObstacleMove)$
if $c_3 \neq \emptyset$ **then**
$p.components \leftarrow [c_1, c_2, c_3]$
$p.cost \leftarrow (|c_1| + |c_3|) * moveCost + |c_2| * pushCost$
$p.minCost \leftarrow |c_2| * pushCost + |c_3| * moveCost$
$p.o, p.d \leftarrow o, d$
$P_{o,d} \leftarrow P_{o,d} \bigcup \{p\}$
end if
end if
end while
return $p \in P_{o,d}$ with minimal $p.cost$ or null if $P_{o,d} = \emptyset$
end procedure

A.2 Interpretation of Levihn's recommandations (as desc. in 3.3)

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
    $\mathcal{O}_{new} \leftarrow \mathcal{O}_{new} \bigcup I.newObstacles$
    $\mathcal{O} \leftarrow I.allObstacles$
    $isPathFree \leftarrow p_{opt} \bigcap \mathcal{O}_{new} \neq \emptyset$
    if not($isPathFree$ 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()
    $c_{next} \leftarrow p_{opt}$.getNextStepComponent()
    $R_{real} \leftarrow$ ROBOT-GOTO(R_{next})
    if $c_{next} = c_2$ 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 $\min(minCostL[i_m].minCost, euCostL[i_e].c_{3_{est}}) < p_{opt}.cost$ **do**
    if $minCostL[i_m].minCost < euCostL[i_e].c_{3_{est}}$ **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
      end if
      $i_e \leftarrow i_e + 1$
    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

A.3 Algorithm adapted to our use case (as desc. in 4.1)

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:        isPathFree ← 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        ▷ [], ≠ and ⋂ operators
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.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(isPathFree 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 35 in Algorithm 3)
37:    end while
38:    return True
39: end procedure
```

Algorithm 7 Obstacle evaluation subroutine taking our hypotheses into account.

procedure PLAN-FOR-OBSTACLE(o, p_{opt}, I, R, R_{goal}, $blockedObsL$)
  if $o \in blockedObsL$ **then**
    return null
  end if
  $P_{o,d} \leftarrow \emptyset$
  for each $pushPose$ in $o.pushPoses$ **do**
    $pushUnit \leftarrow (cos(pushPose.yaw), sin(pushPose.yaw))$ ▷ Unit vector for push direction
    $c_1 \leftarrow A^*(R, pushPose, I.occGrid)$ ▷ c_1 is computed for each push pose
    if $c_1 = \emptyset$ **then**
      continue ▷ **continue**
    end if
    $seq \leftarrow 1$
    $translation \leftarrow pushUnit * onePushDist * seq$ ▷ onePushDist is a distance constant
    $safeSweptArea \leftarrow$GET-SAFE-SWEPT-AREA(o, $translation$, $I.occGrid$)
    $oSimPose \leftarrow pushPose + translation$
    $c3_{(Est)} \leftarrow \{oSimPose, R_{goal}\}$
    $C_{est} \leftarrow (|c_1| + |c3_{(Est)}|) * moveCost + |translation| * o.pushCost$
    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**
        $p.components \leftarrow [c_1, c_2, c_3]$
        $p.cost \leftarrow (|c_1| + |c_3|) * moveCost + |c_2| * o.pushCost$
        $p.minCost \leftarrow |c_2| * o.pushCost + |c_3| * moveCost$
        $p.o \leftarrow$ COPY(o) ▷ COPY
        $p.translation \leftarrow translation$ ▷ Note on saving $translation$ in p_{opt}
        $p.safeSweptArea \leftarrow safeSweptArea$
        $P_{o,d} \leftarrow P_{o,d} \bigcup \{p\}$
        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)
      $oSimPose \leftarrow pushPose + translation$
      $c3_{(Est)} \leftarrow \{oSimPose, R_{goal}\}$
      $C_{est} \leftarrow (|c_1| + |c3_{(Est)}|) * moveCost + |translation| * o.pushCost$
    end while
  end for
  return $p \in P_{o,d}$ with minimal $p.cost$ or null if $P_{o,d} = \emptyset$
end procedure

A.4 Algorithm proposition: Social awareness through manipulation authorization consideration (as desc. in 4.2)

Algorithm 8 Execution loop modified for allowing observation.

1: **procedure** MAKE-AND-EXECUTE-PLAN(R_{init}, R_{goal}, I_{init})
2: ... ▷ Initialization (lines 2 to 5 in Algorithm 6)
3: *isObservable* ← *True*
4: **while** $R \neq R_{goal}$ **do**
5: UPDATE-FROM-NEW-INFORMATION(I)
6: ... ▷ Knowledge update and checks (lines 7 to 29 in Algorithm 6)
7: **if not**(*isPathFree* AND *isManipSuccess* AND *isManipSafe* AND *isPushPoseValid* AND *isObservable*) **then**
8: $p_{opt}.components \leftarrow [A^*(R, R_{goal}, I.occGrid)]$
9: $p_{opt}.cost \leftarrow |p_{opt}| * moveCost$
10: MAKE-PLAN(R, R_{goal}, I, $\mathcal{O}$, *blockedObsL*, p_{opt}, *euCostL*, *minCostL*)
11: $\mathcal{O}_{new} \leftarrow \emptyset$
12: **end if**
13: **if** $p_{opt}.components = \emptyset$ **then**
14: **return** *False*
15: **end if**
16: *isManipSuccess* ← *True*
17: *isObservable* ← *True*
18: $R_{next} \leftarrow p_{opt}$.getNextStep()
19: $c_{next} \leftarrow p_{opt}$.getNextStepComponent()
20: **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**
21: *fObsPose* ← GET-FIRST-PATH-OBSPOSE($p_{opt}.o$, p_{opt}.get-c_0() + p_{opt}.get-c_1(), I)
22: **if** *fObsPose* = null **then**
23: *isObservable* ← *False*
24: **continue**
25: **end if**
26: **end if**
27: $R_{real} \leftarrow$ ROBOT-GOTO(R_{next})
28: **if** $c_{next} = c_2$ AND $R_{real} \neq R_{next}$ **then**
29: *isManipSuccess* ← *False*
30: *blockedObsL* ← *blockedObsL* $\bigcup$ $p_{opt}.o$
31: **end if**
32: $R \leftarrow R_{real}$
33: **end while**
34: **return** *True*
35: **end procedure**

Algorithm 9 Obstacle evaluation subroutine modified for allowing observation.

1: **procedure** PLAN-FOR-OBSTACLE(o, p_{opt}, I, R, R_{goal}, $blockedObsL$)
2: **if** $o \in blockedObsL$ OR $o.movableStatus = IS\_NOT\_MOVABLE$ **then**
3: **return** null
4: **end if**
5: $P_{o,d} \leftarrow \emptyset$
6: **for** each $pushPose$ in $o.pushPoses$ **do**
7: $pushUnit \leftarrow (cos(pushPose.yaw), sin(pushPose.yaw))$
8: $c_1 \leftarrow A^*(R, pushPose, I.occGrid)$
9: **if** $c_1 = \emptyset$ **then**
10: **continue**
11: **end if**
12: $c_0 \leftarrow \emptyset$
13: **if** $o.movableStatus = IS\_MAYBE\_MOVABLE$ **then**
14: $obsPose \leftarrow$ GET-FIRST-PATH-OBSPOSE(o, c_1, I)
15: **if** $obsPose \neq$ null **then**
16: $c_0, c_1 \leftarrow c_1[c_1.firstPose : obsPose], c_1[obsPose : c_1.lastPose]$
17: **else**
18: COMPUTE-C0-C1(o, I, R, $pushPose$, c_0, c_1)
19: **if** $c_0 = \emptyset$ OR $c_1 = \emptyset$ **then**
20: **continue**
21: **end if**
22: **end if**
23: **end if**
24: $seq \leftarrow 1$
25: $translation \leftarrow pushUnit * onePushDist * seq$
26: $safeSweptArea \leftarrow$GET-SAFE-SWEPT-AREA(o, $translation$, I)
27: $oSimPose \leftarrow pushPose + translation$
28: $c3_{(Est)} \leftarrow \{oSimPose, R_{goal}\}$
29: $C_{est} \leftarrow (|(c_0 \neq \emptyset ? |c_0| : 0)| + |c_1| + |c3_{(Est)}|) * moveCost + |translation| * o.pushCost$
30: **while** $C_{est} \leq p_{opt}.cost$ AND $safeSweptArea \neq$ null **do**
31: $c_2 \leftarrow \{pushPose, oSimPose\}$
32: $c_3 \leftarrow A^*(oSimPose, R_{goal}, I.withSimulatedObstacleMove)$
33: **if** $c_3 \neq \emptyset$ **then**
34: $p.components \leftarrow c_0 \neq \emptyset$? $[c_0, c_1, c_2, c_3] : [c_1, c_2, c_3]$
35: $p.cost \leftarrow (|(c_0 \neq \emptyset ? |c_0| : 0)| + |c_1| + |c_3|) * moveCost + |c_2| * o.pushCost$
36: $p.minCost \leftarrow |c_2| * o.pushCost + |c_3| * moveCost$
37: ... ▷ Affectation of other variables of p (lines 25 to 31 in Algorithm 7)
38: **end if**
39: $seq \leftarrow seq + 1$
40: $translation \leftarrow pushUnit * onePushDist * seq$
41: $safeSweptArea \leftarrow$GET-SAFE-SWEPT-AREA(o, $translation$, I)
42: $oSimPose \leftarrow pushPose + translation$
43: $c3_{(Est)} \leftarrow \{oSimPose, R_{goal}\}$
44: $C_{est} \leftarrow (|(c_0 \neq \emptyset ? |c_0| : 0)| + |c_1| + |c3_{(Est)}|) * moveCost + |translation| * o.pushCost$
45: **end while**
46: **end for**
47: **return** $p \in P_{o,d}$ with minimal $p.cost$ or null if $P_{o,d} = \emptyset$
48: **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 11 Subroutine for computing c_0 and c_1 if c_1 is not already valid.

```
1: procedure COMPUTE-C0-C1(o, I, R, pushPose, c_0, c_1)
2:     c_1 ← ∅
3:     totalCost ← +∞
4:     for each obsPose in o.obsPoses do
5:         c_0_cur ← A*(R, obsPose, I.occGrid)
6:         c_1_cur ← A*(obsPose, pushPose, I.occGrid)
7:         newTotalCost = |c_0_cur| + |c_1_cur|
8:         if newTotalCost < +∞ AND (newTotalCost < totalCost OR (newTotalCost =
   totalCost AND |c_0_cur| < |c_0|) then
9:             c_0 = c_0_cur
10:            c_1 = c_1_cur
11:            totalCost = |c_0| + |c_1|
12:        end if
13:    end for
14: end procedure
```

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

procedure OPT-COMPUTE-C0-C1(o, I, R, $pushPose$, c_0, c_1)
    $c_1 \leftarrow \emptyset$
    $totalCost \leftarrow +\infty$
    $euPosesCostL \leftarrow \emptyset$ ▷ Sort observation poses by ascending heuristic cost.
    for each $obsPose$ in $o.obsPoses$ **do**
        $euPosesCostL$.insert($\{obsPose, |R, obsPose| + |obsPose, pushPose|\}$)
    end for
    if $euPosesCostL \neq \emptyset$ **then**
        $op_{next} \leftarrow euPosesCostL[0]$
        while $totalCost \geq op_{next}.cost$ AND $op_{next} \neq$ null **do**
            $c_{0_{cur}} \leftarrow A^*(R, op_{next}.obsPose, I.occGrid)$
            $c_{1_{cur}} \leftarrow A^*(op_{next}.obsPose, pushPose, I.occGrid)$
            $newTotalCost = |c_{0_{cur}}| + |c_{1_{cur}}|$
            if $newTotalCost < +\infty$ AND ($newTotalCost < totalCost$ OR ($newTotalCost = totalCost$ AND $|c_{0_{cur}}| < |c_0|$) **then**
                $c_0 = c_{0_{cur}}$
                $c_1 = c_{1_{cur}}$
                $totalCost = |c_0| + |c_1|$
            end if
            $op_{next} \leftarrow euPosesCostL$.getNext()
        end while
    end if
end procedure

A.5 Algorithm proposition: Social awareness through placement consideration (as desc. in 4.3)

Algorithm 13 Obstacle evaluation subroutine modified for considering placement.

1: **procedure** PLAN-FOR-OBSTACLE(o, p_{opt}, I, R, R_{goal}, $blockedObsL$, $occCostGrid$)
2: ... ▷ Initialization (lines 2 to 4 in Algorithm 7)
3: **for** each $pushPose$ in $o.pushPoses$ **do**
4: ... ▷ Loop initialization (lines 7 to 15 in Algorithm 7)
5: $suppC_M \leftarrow$GET-OCC-COST(GET-OBS-POINTS(o, $translation$), $occCostGrid$)
6: $c3_{(Est)} \leftarrow \{oSimPose, R_{goal}\}$
7: $C_{est} \leftarrow (|c_1| + |c3_{(Est)}|) * moveCost + |translation| * o.pushCost * suppC_M$
8: **while** $C_{est} \leq p_{opt}.cost$ AND $safeSweptArea \neq$ null **do**
9: $c_2 \leftarrow \{pushPose, oSimPose\}$
10: $c_3 \leftarrow A^*(oSimPose, R_{goal}, I.withSimulatedObstacleMove)$
11: **if** $c_3 \neq \emptyset$ **then**
12: $p.components \leftarrow [c_1, c_2, c_3]$
13: $p.cost \leftarrow (|c_1| + |c_3|) * moveCost + |c_2| * o.pushCost * suppC_M$
14: $p.minCost \leftarrow |c_2| * o.pushCost * suppC_M + |c_3| * moveCost$
15: ... ▷ Affectation of other variables of p (lines 25 to 31 in Algorithm 7)
16: **end if**
17: ... ▷ Loop variable update (lines 33 to 36 in Algorithm 7)
18: $suppC_M \leftarrow$GET-OCC-COST(GET-OBS-POINTS(o, $translation$), $occCostGrid$)
19: $c3_{(Est)} \leftarrow \{oSimPose, R_{goal}\}$
20: $C_{est} \leftarrow (|c_1| + |c3_{(Est)}|) * moveCost + |translation| * o.pushCost * suppC_M$
21: **end while**
22: **end for**
23: **return** $p \in P_{o,d}$ with minimal $p.cost$ or null if $P_{o,d} = \emptyset$
24: **end procedure**

Algorithm 14 Obstacle evaluation subroutine modified for considering placement.

1: **procedure** GET-OCC-COST($simOccPoints$, $occCostGrid$)
2: $VALUE\_RANGE \leftarrow (FORBIDDEN\_VALUE - ALLOWED\_VALUE)$
3: $cost \leftarrow 1$
4: **for** each $point$ in $simOccPoints$ **do**
5: $valueForPoint \leftarrow occCostGrid[point]$
6: **if** $valueForPoint = FORBIDDEN\_VALUE$ **then**
7: **return** $+\infty$
8: **else if** $valueForPoint = ALLOWED\_VALUE$ **then**
9: $valueForPoint \leftarrow 0$
10: **end if**
11: $cost \leftarrow cost + valueForPoint / VALUE\_RANGE$
12: **end for**
13: **return** $cost$
14: **end procedure**

A.6 Algorithm proposition: Taking dynamic obstacles into account (as desc. in 4.4)

Algorithm 15 Execution loop taking dynamic obstacles into account.

```
1:  procedure MAKE-AND-EXECUTE-PLAN(R_init, R_goal, I_init)
2:      ...                                   ▷ Initialization (lines 2 to 5 in Algorithm 6)
3:      isBlockingObsMoved ← False
4:      while R ≠ R_goal do
5:          UPDATE-FROM-NEW-INFORMATION(I)
6:          if I.freeSpaceCreated then
7:              minCostL ← ∅
8:          end if
9:          O ← I.allObstacles
10:         isPathFree ← p_opt ⋂ O ≠ ∅
11:         isBlockingObsMoved ← I.movedObstacles ≠ ∅
12:         ...                               ▷ Plan validity checks (lines 14 to 29 in Algorithm 6)
13:         isPlanValid ← (isPathFree AND isManipSuccess AND isManipSafe AND
    isPushPoseValid)
14:         if not isPlanValid then
15:             p_opt.components ← [A*(R, R_goal, I.occGrid)]
16:             p_opt.cost ← |p_opt| ∗ moveCost
17:         end if
18:         if not isPlanValid OR (isBlockingObsMoved) then
19:             MAKE-PLAN(R, R_goal, I, O, blockedObsL, p_opt, euCostL, minCostL)
20:             isManipSuccess ← True         ▷ Line 27 in Algorithm 3 is moved here.
21:             continue
22:         end if
23:         ...                               ▷ Execution (lines 24 to 26 and 28 to 35 in Algorithm 3)
24:         ...                               ▷ Execution (lines 24 to 35 in Algorithm 3)
25:     end while
26:     return True
27: end procedure
```

A.7 Merged proposition algorithm (as desc. in 4.5)

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
    $\mathcal{O}_{new} \leftarrow \mathcal{O}_{new} \bigcup I.newObstacles$
    $\mathcal{O} \leftarrow I.allObstacles$
    $isPathFree \leftarrow p_{opt} \bigcap \mathcal{O}_{new} \neq \emptyset$
    $isBlockingObsMoved \leftarrow I.movedObstacles \neq \emptyset$
    $isPushPoseValid \leftarrow True$
    $isManipSafe \leftarrow True$
    $isObstacleSame \leftarrow True$
    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 $p_{opt}.pushPose \notin p_{opt}.o.pushPoses$ **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
    $isPlanValid \leftarrow$ ($isPathFree$ 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

```
46:         if p_opt.components = ∅ then
47:             return False
48:         end if
49:         isManipSuccess ← True
50:         isObservable ← True
51:         R_next ← p_opt.getNextStep()
52:         c_next ← p_opt.getNextStepComponent()
53:         if p_opt.o ≠ null p_opt.o.movableStatus = IS_MAYBE_MOVABLE AND not
    isObstacleSame AND (c_next = c_0 OR c_next = c_1) then
54:             fObsPose ← GET-FIRST-PATH-OBSPOSE(p_opt.o, p_opt.get-c_0() + p_opt.get-c_1(), I)
55:             if fObsPose = null then
56:                 isObservable ← False
57:                 continue
58:             end if
59:         end if
60:         R_real ← ROBOT-GOTO(R_next)
61:         if c_next = c_2 AND R_real ≠ R_next then
62:             isManipSuccess ← False
63:             blockedObsL ← blockedObsL ⋃ p_opt.o
64:         end if
65:         R ← R_real
66:     end while
67:     return True
68: 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, $translation$), $occCostGrid$)
    $c_{3_{(Est)}} \leftarrow \{oSimPose, R_{goal}\}$
    $C_{est} \leftarrow ((c_0 \neq \emptyset ? |c_0| : 0) + |c_1| + |c_{3_{(Est)}}|) * moveCost + |translation| * o.pushCost * suppC_M$

31: **while** $C_{est} \leq p_{opt}.cost$ AND $safeSweptArea \neq$ null **do**
32: $c_2 \leftarrow \{pushPose, oSimPose\}$
33: $c_3 \leftarrow A^*(oSimPose, R_{goal}, I.withSimulatedObstacleMove)$
34: **if** $c_3 \neq \emptyset$ **then**
35: $p.components \leftarrow c_0 \neq \emptyset$? $[c_0, c_1, c_2, c_3] : [c_1, c_2, c_3]$
36: $p.cost \leftarrow ((c_0 \neq \emptyset ? |c_0| : 0) + |c_1| + |c_3|) * moveCost + |c_2| * o.pushCost * suppC_M$
37: $p.minCost \leftarrow |c_2| * o.pushCost * suppC_M + |c_3| * moveCost$
38: $p.o \leftarrow$ COPY(o)
39: $p.translation \leftarrow translation$
40: $p.safeSweptArea \leftarrow safeSweptArea$
41: $P_{o,d} \leftarrow P_{o,d} \bigcup \{p\}$
42: **if** $p.cost < p_{opt}.cost$ **then**
43: $p_{opt} \leftarrow p$
44: **end if**
45: **end if**
46: $seq \leftarrow seq + 1$
47: $translation \leftarrow pushUnit * onePushDist * seq$
48: $safeSweptArea \leftarrow$GET-SAFE-SWEPT-AREA(o, $translation$, I)
49: $oSimPose \leftarrow pushPose + translation$
50: $suppC_M \leftarrow$GET-OCC-COST(GET-OBS-POINTS(o, $translation$), $occCostGrid$)
51: $c_{3_{(Est)}} \leftarrow \{oSimPose, R_{goal}\}$
52: $C_{est} \leftarrow ((c_0 \neq \emptyset ? |c_0| : 0) + |c_1| + |c_{3_{(Est)}}|) * moveCost + |translation| * o.pushCost * suppC_M$
53: **end while**
54: **end for**
55: **return** $p \in P_{o,d}$ with minimal $p.cost$ or null if $P_{o,d} = \emptyset$
56: **end procedure**

A.8 Efficient Opening Detection, Levihn M. and Stilman M. (2011), Commented

Note on GET-M_i'-MATRIX The world W is represented by an occupancy grid: this procedure extends the obstacle M_i by the robot's diameter, giving us M_i', then represented as a binary matrix M, which has the size of the bounding box of M_i'.

Note on GET-NEW-X/Y-POS Simulate the set of manipulation actions A_M and get the world coordinates of the object.

Note on interpreting the value of Z If Z is the 0-matrix, it means no new openings were detected, as all blocking areas still are blocking after the manipulation. Else, it means that one intersecting area has disappeared, meaning a possible new opening.

Note on detecting BAs Check for blockage between M_i' (represented by M) and other obstacles (which data is contained in the occupancy grid G). For that we only call ASSIGN-NR if at least one of the two current elements of both matrices signals an obstacle ($\neq 0$).

Note on deletion If an index has already been deleted in an element of BS, delete it everywhere else because if part of a previous blocking area is detected, it means that the robot is still blocked by the same area. If for a same element of BS, there is an index in $BA[x][y] and BA_s^*[x][y]$, then it means that the blocking area still exists, thus we zero it in BS.

Algorithm 18 Efficient Local Opening Detection algorithm, Levihn et. al. (2011), commented

procedure CHECK-NEW-OPENING(G, M_i, A_M, BA)
  $M \leftarrow$ GET-M_i'-MATRIX(M_i) ▷ Note on GET-M_i'-MATRIX
  $x_{offset} \leftarrow M_i.x$ ▷ $M_i.x$ and $M_i.y$ are the map coordinates of M_i's top left corner.
  $y_{offset} \leftarrow M_i.y$
  if BA is null **then** ▷ BA needs not be recomputed if the environment did not change.
    $BA \leftarrow$ GET-BLOCKING-AREAS(x_{offset}, y_{offset}, M, G) ▷ Blocking areas before manip.
  end if
  $x_{offset} \leftarrow$ GET-NEW-X-POS(A_M, M_i) ▷ Note on GET-NEW-X/Y-POS
  $y_{offset} \leftarrow$ GET-NEW-Y-POS(A_M, M_i)
  $BA_s \leftarrow$ GET-BLOCKING-AREAS(x_{offset}, y_{offset}, M, G) ▷ Blocking areas after manip.
  $BA_s^* \leftarrow [0][0](dim(M))$ ▷ Since the window of BA_s shifted with the manipulation of M_i, ...
  for k from 0 to $|BA_s^*|$ **do** ▷ ... we shift it back for the future comparison with BA.
    for l from 0 to $|BA_s^*[i]|$ **do**
      $x \leftarrow (x_{offset} - M_i.x) + k$
      $y \leftarrow (y_{offset} - M_i.y) + l$
      if $0 < x < |BA_s^*|$ AND $0 < y < |BA_s^*[x]|$ **then**
        $BA_s^*[x][y] \leftarrow |BA_s^*|[k][l]$
      end if
    end for
  end for
  $Z \leftarrow$ COMPARE(BA, BA_s^*) ▷ Finally, compare the two blocking aread configurations.
  if $Z = [0][0](dim(M))$ **then** ▷ Note on interpreting the value of Z
    return $false$
  end if
  return $true$
end procedure

procedure GET-BLOCKING-AREAS(x_{off}, y_{off}, M, G)
  $index \leftarrow 1$
  $BA \leftarrow [0][0](dim(M))$ ▷ $[0][0](dim(M))$ represents the 0-Matrix of dimensions = $dim(M)$
  for x from 0 to $|M|$ **do** ▷ Iterate over M to detect and tag blocking areas.
    for y from 0 to $|M[x]|$ **do**
      if $M[x][y] \neq 0$ AND $G[x + x_{off}][y + y_{off}] \neq 0$ **then** ▷ Note on detecting BAs
        ASSIGN-NR(BA, x, y, $index$) ▷ BA and $index$ are directly modified in the call.
      end if
    end for
  end for
  return BA ▷ Return the saved information on the blocked areas.
end procedure

```
40: procedure ASSIGN-NR(BA, x, y, index)
41:     for i from -1 to 1 do          ▷ Assignment is performed based on the 3*3 neighborhood.
42:         for j from -1 to 1 do
43:             if BA[x + i][y + j] ≠ 0 then          ▷ If a number is already in the neighborhood, ...
44:                 BA[x][y] ← BA[x + i][y + j]       ▷ ... the same number is assigned to the element.
45:                 return
46:             end if
47:         end for
48:     end for
49:     BA[x][y] ← index              ▷ Else a new number is assigned, equal to the new index.
50:     index ← index + 1             ▷ Only increment index if new intersection is created.
51:     return
52: end procedure
53:
54: procedure COMPARE(BA, BA*_s)       ▷ This function checks for non-zero entries in both
    matrices.
55:     BS ← COPY(BA)
56:     del_num ← ∅                   ▷ Set of the indexes of obstacles to delete from BS.
57:     for x from 0 to |BS| do                                  ▷ Iterate over BS.
58:         for y from 0 to |BS[x]| do
59:             if BA[x][y] ∈ del_num then                       ▷ Note on deletion
60:                 BS[x][y] ← 0
61:             end if
62:             if BA[x][y] ≠ 0 AND BA*_s[x][y] ≠ 0 then         ▷ Note on deletion
63:                 del_num = del_num ⋃ BS[x][y]
64:                 BS[x][y] ← 0
65:             end if
66:         end for
67:     end for
68:     return BS
69: end procedure
```

Appendix B

Comparison tables

B.1 Detailed comparison tables

Year	2004	2005	2007	2008
Authors	Okada, K.; Haneda, A.; Nakai, H.; Inaba, M.; Inoue, H.	Stilman, Mike; Kuffner, James J.	Stilman, Mike; Nishiwaki, Koichi; Kagami, Satoshi; Kuffner, James J.	Stilman, Mike; Kuffner, James
Title	Environment manipulation planner for humanoid robots using task graph that generates action sequence	Navigation among movable obstacles: real-time reasoning in complex environments	Planning and executing navigation among movable obstacles	Planning Among Movable Obstacles with Artificial Constraints
Conference / Journal	2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)	International Journal of Humanoid Robotics	Advanced Robotics	The International Journal of Robotics Research
Filename (bibtex id)	okada_environment_2004	stilman_navigation_2005	stilman_planning_2007	stilman_planning_2008
Reference	[a]	[b]	[c]	[d]
Hypotheses				
Knowledge of the environment	3D metric map, complete with perfect data.	2D metric map, complete with perfect data.	3D metric map, partial with nearly perfect data. Object poses are estimated at 30 Hzb a global tracking system using markers and infrared cameras. This system also tracks the robot, that brings extra data with its embedded encoders and force sensors.	2D metric map, complete with perfect data.
Obstacle characteristics	Walls, tables, chairs with wheelcasters, cardboard boxes and trash cans. Simplified into rectangular cuboids. No autonomously moving obstacles. Path planning done on a 2D projection from above. Movable obstacles have extra semantic data associated with them: “grasping points”, weight et a pre-registered procedure to move them. Obstacles can be moved in 2D configuration space, in translation and rotation movements.	Walls, round and rectangular tables, loveseats, sofas and chairs. No autonomously moving obstacles. Path planning done with the 2D metric map. Movable obstacles have extra semantic data associated with them: mass center, mass and center of mass. Obstacles can be moved in 2D configuration space, in translation and rotation movements.	Walls, rectangular tables and chairs with wheelcasters. No autonomously moving obstacles. Path planning done on a 2D projection from above with a computation of the convex hull of the 3D mesh. Movable obstacles have extra semantic data associated with them: “grasping points”, mass et center of mass. Obstacles can be moved in 2D configuration space, in translation only, and for heavy objects, only according to the perpendicular axis to the contact line.	Walls, round and rectangular tables, loveseats, sofas and chairs. No autonomously moving obstacles. Path planning done on a 2D occupancy grid, rasterized from the 2D metric map with convex hull of obstacle geometric representation. Movable obstacles have extra semantic data associated with them: mass, center of mass and inertial moment. Obstacles can be moved in 2D configuration space, in translation and rotation movements.
Robot characteristics	Simulated HRP2 Robot (characteristics: http://global.kawada.jp/mechatronics/hrp2.html) with unlimited field of vision. ECan move in a 2D configuration space, in translation and rotation. The robot can lift & drop obstacles.	Nondescript simulated humanoïd robot with unlimited field of vision. Can move in a 2D configuration space, in translation and rotation. The robot can push & pull obstacles.	Real HRP2 Robot (characteristics: http://global.kawada.jp/mechatronics/hrp2.html) with unlimited field of vision obtained through global tracking systeme path planning. Can move in a 2D configuration space, in translation and rotation. The robot can push & pull obstacles.	Same as stilman_navigation_2005.
Problem class	Not explicit but probably a subset of L1.	L1.	L1.	LkM.

Year	2010	2010	2013	2013
Authors	Wu, Hai-Ning; Levihn, M.; Stilman, M.	Kakiuchi, Y.; Ueda, R.; Kobayashi, K.; Okada, K.; Inaba, M.	Levihn, M.; Kaelbling, L. P.; Lozano-Pérez, T.; Stilman, M.	Levihn, M.; Scholz, J.; Stilman, M.
Title	Navigation Among Movable Obstacles in unknown environments	Working with movable obstacles using on-line environment perception reconstruction using active sensing and color range sensor	Foresight and reconsideration in hierarchical planning and execution	Planning with movable obstacles in continuous environments with uncertain dynamics
Conference / Journal	2010 IEEE/RSJ International Conference on Intelligent Robots and Systems	2010 IEEE/RSJ International Conference on Intelligent Robots and Systems	2013 IEEE/RSJ International Conference on Intelligent Robots and Systems	2013 IEEE International Conference on Robotics and Automation
Filename (bibtex id)	wu_navigation_2010	kakiuchi_working_2010	levihn_foresight_2013	levihn_planning_2013
Reference	[e]	[f]	[g]	[h]
Hypotheses				
Knowledge of the environment	2D metric map, unknown with perfect data. Hypothesis of unknown space is free space.	3D metric map, unknown with approximative data. The environment configuration is obtained only through onboard sensors. Hypothesis of unknown space is free space.	3D metric map, partial with approximative data (unmovable objects known, movable objects unknown).	Non-discretized 2D metric map, complete with approximative data.
Obstacle characteristics	Rectangular cuboids. No autonomously moving obstacles. No extra semantic data on obstacles. Path planning done on a 2D occupancy grid, rasterized from the 2D metric map. Obstacles can only be moved in 2D configuration space, in translation only along the axes of the plane.	Walls, static rectangular tables and chairs with wheelcasters. No autonomously moving obstacles. No extra semantic data on obstacles. Path planning done on a 2D projection from above with a computation of the convex hull of the 3D mesh that is previously reduced to a prism. Obstacles can be moved in 2D configuration space, in translation only and in a single direction.	Walls, static rectangular tables, carboard boxes and chairs with wheelcasters. No autonomously moving obstacles. No extra semantic data on obstacles. Path planning done on a 2D projection from above. Obstacles can be moved in 2D configuration space, in translation and rotation movements.	Walls, round tables, rectangular loveseats. No autonomously moving obstacles. Path planning done on the 2D non-discretized map. Movable obstacles have extra semantic data associated with them: mass, center of mass, cinematics or frictions (which are determined online through manipulation). Obstacles can be moved in 2D configuration space, in translation and rotation movements.
Robot characteristics	Nondescript simulated wheeled robot with a limited field of vision. Can move in a 2D configuration space, in translation and rotation. The robot can only push obstacles.	Real HRP2 Robot (characteristics: http://global.kawada.jp/mechatronics/hrp2.html) with onboard limited field of vision (Swissranger SR-400: http://www.realtechsupport.org/UB/SR/range_finding/SR4000_SR4500_Manual.pdf and Pointgray Flea2: https://eu.ptgrey.com/support/downloads/10117). Can move in a 2D configuration space, in translation and rotation. The robot can push obstacles.	PR2 Robot (characteritics: http://www.willowgarage.com/pages/pr2/specs) with onboard limited field of vision (Microsoft Kinect V1). Can move in a 2D configuration space, in translation and rotation. The robot can push obstacles.	Simulated humanoïd robot GOLEM KRANG (characteristics: http://www.golems.org/projects/krang.html) with unlimited field of vision. Can move in a 2D configuration space, in translation and rotation. The robot can pull or push obstacles.
Problem class	Subset of L1 since a plan can only contain the manipulation of one obstacle.	Subset of L1 : the robot does not seek to move an obstacle if it can reach its goal without manipulating any obstacle.	Not explicit but probably LkM according to the explanations.	Not explicit but probably L1 according to the explanations.

Year	2014	2014	2015	2016
Authors	Levihn, M.; Stilman, M.; Christensen, H.	Clingerman, C.; Lee, D. D.	Clingerman, C.; Wei, P. J.; Lee, D. D.	Scholz, J.; Jindal, N.; Levihn, M.; Isbell, C. L.; Christensen, H. I.
Title	Locally optimal navigation among movable obstacles in unknown environments	Estimating manipulability of unknown obstacles for navigation in indoor environments	Dynamic and probabilistic estimation of manipulable obstacles for indoor navigation	Navigation Among Movable Obstacles with learned dynamic constraints
Conference / Journal	2014 IEEE-RAS International Conference on Humanoid Robots	2014 IEEE International Conference on Robotics and Automation (ICRA)	2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)	2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Filename (bibtex id)	levihn_locally_2014	clingerman_estimating_2014	clingerman_dynamic_2015	scholz_navigation_2016
Reference	[i]	[j]	[k]	[l]
Hypotheses				
Knowledge of the environment	2D metric map, unknown with perfect data. Hypothesis of unknown space is free space.	2D costmap, unknown with approximative data. Hypothesis of unknown space is free space.	Same as clingerman_estimating_2014, but the map representation in memory grows only when new areas are explored to gain computing performance.	Non-discretized 2D metric map, complete with approximative data.
Obstacle characteristics	Walls, round tables, rectangular loveseats. No autonomously moving obstacles. No extra semantic data on obstacles. Obstacles can be moved in 2D configuration space, in translation only in a single direction.	Walls, static heavy cardboard boxes, chair with wheelcasters. No autonomously moving obstacles. No extra semantic data on obstacles. Path planning done on the 2D discretized costmap. Obstacles can be moved in 2D configuration space, in translation only in a single direction.	Same as clingerman_estimating_2014 but the use of D*Lite allows to take autonomously moving obstacles into account.	Same as scholz_navigation_2016.
Robot characteristics	Nondescript simulated wheeled robot with a limited field of vision. Can move in a 2D configuration space, in translation and rotation. The robot can push & pull obstacles.	Custom robot vehicle for MAGIC 2010 Competition, assimilable to an RC car, with a limited conic field of vision obtained through the fusion of data between a RGB camera and a LIDAR projecting to a distance of up to 10m. An embedded IMU and wheel encoders help localize the robot. A front bumper is used to push the obstacles. Can move in a 2D configuration space, in translation and rotation. The robot can push obstacles.	Same as clingerman_estimating_2014.	Real humanoïd robot GOLEM KRANG (characteristics: http://www.golems.org/projects/krang.html) with unlimited field of vision obtained through an external camera positioning system. Can move in a 2D configuration space, in translation and rotation. The robot can pull or push obstacles.
Problem class	Subset of L1 since a plan can only contain the manipulation of one obstacle.	Not explicit but probably a subset of L1.	Same as clingerman_estimating_2014.	Not explicit but probably L1 according to the explanations.

Filename (bibtex id)	okada_environment_2004	stilman_navigation_2005	stilman_planning_2007	stilman_planning_2008
Reference	[a]	[b]	[c]	[d]
Approaches				
Path Planning Algorithm(s) and heuristics	No explicit mention.	A* subroutine. A motion planner heuristic is introduced to improve performance but is not admissible (only "well-informed"). Use a grid-search type subroutine that considers collisions as soft constraints to find disjoint regions and obstacles to move in priority.	Same as stilman_navigation_2005.	For a transit path (no obstacle manipulation), A* is used with a euclidean heuristic augmented with a single penalty if the path penetrates the configuration space of the previously artificially constrained obstacle. For a transit path (with obstacle manipulation), a BFS algorithm is used with a heuristic that penalizes the penetration of a movable obstacle's configuration space.
Evaluation and evolution of an obstacle's "movable" characteristic and its associated cost	No online evaluation of the "movable" characteritic of an obstacle, this is already known since the beginning. The cost of moving an obstacle is defined as the energy necessary to move its weigth.	No online evaluation of the "movable" characteritic of an obstacle, this is already known since the beginning. The cost of moving an obstacle is defined as the energy necessary to move its weigth.	Same as stilman_navigation_2005.	No online evaluation of the "movable" characteritic of an obstacle, this is already known since the beginning. No particular cost of moving the obstacle except the cost of the movement itself.
Object manipulation maneuver planning	The kinematic/friction constraints of the object are not taken into account. The placement of the robot explicitly depends on pre-determined "grasping points" that depend on the obstacle geometric configuration.	The kinematic/friction constraints of the object are not taken into account. The placement of the robot explicitly depends on pre-determined "contact points" that depend on the obstacle geometric configuration.	The friction constraints of the object are mildly taken into account by locally adaptating the object's prehension and the robot's posture to keep it on the expected trajectory. The placement of the robot explicitly depends on pre-determined "contact points" that depend on the obstacle geometric configuration.	The kinematic/friction constraints of the object are not taken into account. The placement of the robot explicitly depends on pre-determined "contact points" that depend on the obstacle geometric configuration.
Planning taking uncertainty into account	None.	None.	Uncertainty is managed through continuous verification of confimity to the generated plan and replanning are triggered whenever necessary. The approach and grasping procedures are progressive and allow to adjust the robot pose locally to improve the chances of successful manipulation.	None.
Performance criteria				
Evaluation in a simulated/real setting	Simulation.	Simulation.	Real.	Simulation.
Computation time	No performance statements : will suppose that is not usable in real-time.	Usable in real-time if heuristic is used.	Usable in real-time.	Usable in real-time.
Optimality and completeness	No guaranteed optimality. No guaranteed completeness.	Guaranteed global optimality if heuristic is not used. Guaranteed completeness in any case.	No guaranteed optimality. Guaranteed completeness.	No guaranteed optimality. No guaranteed completeness.
Optimality type	Distance or energy.	Number of moved obstacles and energy.	Energy.	Distance and minimal traversal of the movable obstacles configuration spaces.
Social acceptability	No interactions with human beings nor consideration of social norms.	Mentions the possibility of taking into account the risk of moving an obstacle because of its fragility, but does not propose anything to deal with this.	No interactions with human beings nor consideration of social norms.	No interactions with human beings nor consideration of social norms.
Number and Density of obstacles	No statements. May suppose a maximal number of movable obstacles < 10 from the figures.	Maximal tested number of movable obstacles $= 90$.	Maximal tested number of movable obstacles $= 10$.	Maximal tested number of movable obstacles $= 9$.

Filename (bibtex id)	wu_navigation_2010	kakiuchi_working_2010	levihn_foresight_2013	levihn_planning_2013
Reference	[e]	[f]	[g]	[h]
Approaches				
Path Planning Algorithm(s) and heuristics	A* path finding subroutine with a euclidean distance heuristic. The overall algorithm also uses a heuristic to determine the order in which to evaluate obstacles and another to stop plan evaluations when no lower cost plan including a specific obstacle can be found. One of the heuristic destroys optimality.	RRT (Rapidly exploring Random Tree) path finding algorithm. No particular heuristic is mentioned.	RRT (Rapidly exploring Random Tree) path finding algorithm. No particular heuristic is mentioned. Use of a “peephole optimization” method to execute the elements of a computed plan in a more efficient way.	Uses RRT variations: KDRRT (Kinodynamic-RRT) and FPRRT (Low-Dimensional RRT). No particular heuristic is mentioned.
Evaluation and evolution of an obstacle’s "movable" characteristic and its associated cost	An obstacle is considered movable until a manipulation fails, it is then considered blocked. The cost of moving an obstacle is a pre-determined constant multiplied by the manipulation path length : depending on the dimension of the constant, this computation could be assimilated to an energy or a time.	An obstacle is considered movable until a manipulation fails, it is then considered blocked. No particular cost of moving the obstacle except the cost of the movement itself.	All initially known obstacles are tagged as static. Any detected obstacle is identified through computer vision during navigation and is deemed movable or not then. No particular cost of moving the obstacle except the cost of the movement itself.	An obstacle is considered movable until a manipulation fails, it is then considered blocked.
Object manipulation maneuver planning	The kinematic/friction constraints of the object are not taken into account. The placement of the robot does not explicitly depend on pre-determined “contact points”, but the figures and experimental video show that such points are used in the implementation because the robot systematically enters in contact with the center of the manipulated obstacle’s side.	The kinematic/friction constraints of the object are not taken into account. The placement of the robot does not depend on pre-determined but on dynamically computed “grasping points”, that are situated at the middle of the top sides of the moved object.	The kinematic/friction constraints of the object are not taken into account. The placement of the robot does not explicitly depend on pre-determined “contact points”, but it can be infered from the experimental video material that they actually use them.	The kinematic/friction constraints are taken into account, which is one of the main points of the algorithm. The placement of the robot explicitly depends on pre-determined “grasping poses” that depend on the obstacle geometric configuration.
Planning taking uncertainty into account	None.	A probabilistic model is used to determine the configuration of obstacles from the perceived point cloud. Use of an algorithm named “color-ICP” to estimate the movement of the obstacle as it is moved by the robot.	Use of an “Unscented Kalman Filter” to estimate the relative poses of the robot to the objects, and the unknown space is actually treated differently than free space through a “war fog”: the algorithm actually checks if it has sufficiently observed the environment in order to act. Also use a probabilistic technique of “e-shadows” to associate a heuristic cost of traversing a zone near an obstacle. Waypoints are distributed in the map to reduce positioning error .	Use of PRM (Probabilistic RoadMaps) to create a navigation subgraph for every free-space zone. A Markov Decision Process is created from the PRM, to manage the possibility of manipulation failures. Monte Carlo simulations in physics engine allow to estimate the success probabilities of a manipulation action.
Performance criteria				
Evaluation in a simulated/real setting	Simulation.	Real.	Real.	Simulation.
Computation time	Usable in real-time.	Usable in real-time.	Usable in real-time.	Not usable in real-time.
Optimality and completeness	No guaranteed optimality. No guaranteed completeness.	No guaranteed optimality. No guaranteed completeness.	No guaranteed optimality but has been improved compared to previous implementations of BHPN. No guaranteed completeness.	Guaranteed optimality with error epsilon. Guaranteed completeness if epsilon = 0.
Optimality type	Energy.	Distance and minimal number of moved obstacles.	Probability of reaching the goal and Distance.	Time, energy and probabiblity of succeding in a manipulation.
Social acceptability	No interactions with human beings nor consideration of social norms.	Mentions the possibility of taking into account the risk of moving an obstacle because of its fragility, but does not propose anything to deal with this.	No interactions with human beings nor consideration of social norms.	No interactions with human beings nor consideration of social norms.
Number and Density of obstacles	Maximal tested number of movable obstacles = 20.	Maximal tested number of movable obstacles = 3.	Maximal tested number of movable obstacles = 14.	Maximal tested number of movable obstacles = 30.

Filename (bibtex id)	levihn_locally_2014	clingerman_estimating_2014	clingerman_dynamic_2015	scholz_navigation_2016
Reference	[i]	[j]	[k]	[l]
Approaches				
Path Planning Algorithm(s) and heuristics	Uses a D* Lite path finding subroutine. The overall algorithm also uses a heuristic to determine the order in which to evaluate obstacles and another to stop plan evaluations when no lower cost plan including a specific obstacle can be found. The heuristics no longer destroy optimality.	ARA* algorithm with a euclidean distance heuristic. Also uses a Lower Confidence Bound (LCB) instead of a heuristic sum to make the algorithm more “exploratory”.	D-Lite* algorithm with a euclidean distance heuristic. Also uses a Lower Confidence Bound (LCB) instead of a heuristic sum to make the algorithm more “exploratory”.	RRT + model-dependent manipulation policy.
Evaluation and evolution of an obstacle’s "movable" characteristic and its associated cost	An obstacle is considered movable until a manipulation fails, it is then considered blocked. The cost of moving an obstacle is a pre-determined constant multiplied by the manipulation path length : depending on the dimension of the constant, this computation could be assimilated to an energy or a time.	The estimation of the cost of manipulating an obstacle is done while navigating and is based off the results tentative interaction to obstacles with similat visual features. This is translated by a cost random variable that has a normal distribution. By default, at the beginning of the experiment (before any sort of learning), any obstacle is considered potentially movable (even walls). If through interaction, an obstacle is deemed movable in a specific direction, it is supposed to be movable in any direction. The cost values depend on a ratio between measured reverse speed and expected reverse speed.	Same as clingerman_estimating_2014, but the random variable has a gamma distribution.	Same as scholz_navigation_2016.
Object manipulation maneuver planning	The kinematic/friction constraints of the object are not taken into account. The placement of the robot does not explicitly depend on pre-determined “contact points”, but the figures and experimental video show that such points are used in the implementation because the robot systematically enters in contact with the center of the manipulated obstacle’s side.	The kinematic/friction constraints of the object are not taken into account. The placement of the robot does not depend on “contact points”.	Same as clingerman_estimating_2014.	The kinematic/friction constraints are taken into account, which is one of the main points of the algorithm. The placement of the robot explicitly depends on pre-determined “grasping points” that depend on the obstacle geometric configuration.
Planning taking uncertainty into account	None.	The occupancy grid costmap integrates the notion of uncertainty directly. Probabilistic models are used to map the RGB sensors date with with the movability status of a cell. A Kalman filter is also used to update the cost distribution associated with a cell. Regular pauses in the robot’s movement allow for positioning recalibration.	Same as clingerman_estimating_2014.	Same as scholz_navigation_2016 + PBRL (Physics-Based Reinforcment Learning) to manage a great variety of manipulation cases.
Performance criteria				
Evaluation in a simulated/real setting	Simulation.	Real.	Same as clingerman_estimating_2014.	Real.
Computation time	Usable in real-time.	Usable in real-time.	Same as clingerman_estimating_2014.	Usable in real-time.
Optimality and completeness	Guaranteed local optimality. No guaranteed completeness.	No guaranteed optimality. No guaranteed completeness.	Same as clingerman_estimating_2014.	Same as scholz_navigation_2016.
Optimality type	Energy.	Time.	Fusion between distance, time and rotation cost.	Time, force and moment.
Social acceptability	No interactions with human beings nor consideration of social norms.	No interactions with human beings nor consideration of social norms.	Same as clingerman_estimating_2014.	Same as scholz_navigation_2016.
Number and Density of obstacles	Maximal tested number of movable obstacles = 70.	Maximal tested number of movable obstacles = 3.	Same as clingerman_estimating_2014.	Maximal tested number of movable obstacles = 2.

B.2 Cross-comparison tables

CROSS COMPARISON TABLE BETWEEN HYPOTHESES AND APPROACHES

			Path Planning Algorithm(s) and heuristics								Evaluation and evolution of an obstacle's "movable" characteristic and its associated cost					Object manipulation maneuver planning			Planning taking uncertainty into account					
			A*	ARA*	D* Lite	BFS	RRT	Standard Heuristic for Path Planning	Custom Heuristic for Path Planning	Supplementary Heuristics	"Movability" (re)evaluated on runtime	Manipulation cost depends on the obstacle's physics metadata	Manipulation cost depends on a constant common to all obstacles	Cost is estimated on runtime	Cost is pre-estimated by a heuristic	Kinematic/Friction constraints taken into account	Limited grasping points number	No concern about grasping points	Adaptive obstacle approach procedures	Use of a Kalman filter	Use of e-shadows	Use of PRM + MDP + MonteCarlo	Use of PBRL	Pointcloud correction
Knowledge of the environment	2D metric map	***[b], [d], [e],[h], [i], [l], [Exp]***	***[b] [d] [e] [Exp]***		[i]	[d]	[h] [l]	***[b] [e] [i] [Exp]***	[d]	***[b] [d] [e] [i] [l] [Exp]***	***[e] [h] [i] [l] [Exp]***	[b]	***[e] [i] [Exp]***		[i]	[h] [l]	***[b] [d] [e] [h] [l] [Exp]***		[h] [l]			[h] [l]	[l]	
	2D costmap	[j], [k]		[j]	[k]			[j]		[j] [k]	[j] [k]			[j] [k]				[j] [k]	[j] [k]	[j] [k]				
	3D metric map	[a], [c], [f], [g]	[c]				[f] [g]	[c]		[c] [g]	[f] [g]	[a] [c]				[c]	[a] [c] [f] [g]		[c] [g]	[g]	[g]			[f]
	Complete	[a], [b], [d], [h], [l]	[b] [d]			[d]	[h] [l]	[b]	[d]	[b] [d] [l]	[h] [l]	[a] [b]				[h] [l]	[a] [h] [d] [h] [l]		[h] [l]			[h] [l]	[l]	
	Partial	***[c], [g], [Exp]***	***[c] [Exp]***				[g]	***[c] [Exp]***		***[c] [g] [Exp]***	***[g] [Exp]***	[c]	***[Exp]***			[c]	***[c] [g] [Exp]***		[c] [g]	[g]	[g]			
	Unknown	[e], [i], [f], [j], [k]	[e]	[j]	[i] [k]		[f]	[e] [i] [j]		[e] [i] [j] [k]	[e] [f] [i] [j] [k]		[e] [i]	[j] [k]	[i]		[e] [f]	[j] [k]	[j] [k]	[j] [k]				[f]
	Perfect data	***[a], [b], [d], [e], [i], [Exp]***	***[b] [d] [e] [Exp]***		[i]	[d]		***[b] [e] [i] [Exp]***	[d]	***[b] [d] [e] [i] [Exp]***	***[e] [i] [Exp]***	[a] [b]	***[e] [i] [Exp]***		[i]		***[e] [b] [d] [e] [Exp]***							
	Approximative data	[c], [f], [g], [h], [j], [k], [l]	[c]	[j]	[k]		[f] [g] [h] [l]	[c] [j]		[c] [g] [j] [k] [l]	[f] [g] [h] [j] [k] [l]	[c]		[j] [k]		[c] [h] [l]	[c] [f] [g] [h] [l]	[j] [k]	[c] [g] [h] [j] [k] [l]	[g] [j] [k]	[g]	[h] [l]	[l]	[f]
	Free unknown space hypothesis	***[e], [f], [i], [j], [k], [Exp]***	***[e] [Exp]***	[j]	[i] [k]		[f]	***[e] [i] [j] [Exp]***		***[e] [i] [j] [k] [Exp]***	***[e] [f] [i] [j] [k] [Exp]***		***[e] [i] [Exp]***	[j] [k]	[i]		***[e] [f] [Exp]***	[j] [k]	[j] [k]	[j] [k]				[f]
Obstacle characteristics	Naive 2D projection	[a], [b], [e], [g], [h], [i], [l]	[b] [e]		[i]		[g] [h] [l]	[h] [e] [i]		[b] [e] [g] [i] [l]	[e] [g] [h] [i] [l]	[a] [b]	[e] [i]		[i]	[h] [l]	[a] [b] [e] [g] [h] [l]		[g] [h] [l]	[g]	[g]	[h] [l]	[l]	
	2D Projection using Convex-Hull	***[c], [d], [f], [Exp]***	***[c] [d] [Exp]***			[d]	[f]	***[c] [Exp]***	[d]	***[c] [d] [Exp]***	***[f] [Exp]***	[c]	***[Exp]***			[c]	***[c] [d] [f] [Exp]***		[c]					[f]
	Any obstacle types	[f], [g], [h], [i], [j], [k], [l]		[j]	[i] [k]		[f] [g] [h] [l]	[i] [j]		[g] [i] [j] [k] [l]	[f] [g] [h] [i] [j] [k] [l]		[i]	[j] [k]	[i]	[h] [l]	[f] [g] [h] [l]	[j] [k]	[g] [h] [j] [k] [i]	[g] [j] [k]	[g]	[h] [l]	[l]	[f]
	Only polygonal obstacles	***[a], [b], [c], [d], [Exp]***	***[b] [c] [d] [Exp]***			[d]		***[b] [c] [Exp]***	[d]	***[b] [c] [d] [Exp]***	***[Exp]***	[a] [b] [c]	***[Exp]***			[c]	***[a] [b] [c] [d] [Exp]***		[c]					
	Only rectangular obstacles	[e]	[e]					[e]		[e]	[e]		[e]				[e]							
	Human obstacle																							
	Moving obstacle	***[k], [Exp]***	***[Exp]***		[k]			***[k] [Exp]***		***[k] [Exp]***	***[k] [Exp]***		***[Exp]***	[k]			***[Exp]***	[k]	[k]	[k]				
	Metadata on obstacle's physics	[a], [b], [c], [d], [h]	[b] [c] [d]			[d]	[h]	[b] [c]	[d]	[b] [c] [d]	[h]	[a] [b] [c]				[c] [h]	[a] [b] [c] [d] [h]		[c] [h]			[h]		
	Obstacle can be translated in 2D plane	***[a], [b], [c], [d], [e], [f], [g], [a], [i], [j], [k], [l], [Exp]***	***[b] [c] [d] [e] [Exp]***	[j]	[i] [k]	[d]	[f] [g] [h] [l]	***[b] [c] [e] [i] [j] [Exp]***	[d]	***[b] [c] [d] [e] [g] [i] [j] [k] [l] [Exp]***	***[e] [f] [g] [h] [i] [j] [k] [l] [Exp]***	[a] [b] [c]	***[e] [i] [Exp]***	[j] [k]	[i]	[c] [h] [l]	***[a] [b] [c] [d] [e] [f] [g] [h] [i] [Exp]***	[j] [k]	[c] [g] [h] [j] [k] [l]	[g] [j] [k]	[g]	[h] [l]	[l]	[f]
	Translation limited to the 2D plane axes	[e]	[e]					[e]		[e]	[e]		[e]				[e]							
	Obstacle can be rotated in the normal to the 2D plane	[a], [b], [d], [g], [h], [l]	[b] [d]			[d]	[g] [h] [l]	[b]	[d]	[b] [d] [g] [l]	[g] [h] [l]	[a] [b]				[h] [l]	[a] [b] [d] [g] [h] [l]		[g] [h] [l]	[g]	[g]	[h] [l]	[l]	
Robot characteristics	HRP2 Robot	[a], [c], [f]	[c]				[f]	[c]		[c]	[f]	[a] [c]				[c]	[a] [c] [f]		[c]					[f]
	PR2 Robot	[g]					[g]			[g]	[g]						[g]		[g]	[g]	[g]			
	GOLEM Krang Robot	[h], [l]					[h] [l]			[l]	[h] [l]					[h] [l]	[h] [l]		[h] [l]			[h] [l]	[l]	
	Custom robot vehicle for MAGIC 2010 Competition	[j], [k]		[j]	[k]			[j]		[j] [k]	[j] [k]			[j] [k]				[j] [k]	[j] [k]	[j] [k]				
	Pepper Robot	***[Exp]***	***[Exp]***					***[Exp]***		***[Exp]***	***[Exp]***		***[Exp]***				***[Exp]***							
	Nondescript humanoid robot	[b], [d]	[b] [d]			[d]		[b]	[d]	[b] [d]		[b]					[b] [d]							
	Nondescript wheeled robot	[e], [i]	[e]		[i]			[e] [i]		[e] [i]	[e] [i]		[e] [i]		[i]		[e]							
	Limited field of vision	***[e], [f], [g], [i], [j], [k], [Exp]***	***[e] [Exp]***	[j]	[i] [k]		[f] [g]	***[e] [i] [j] [Exp]***		***[e] [g] [i] [j] [k] [Exp]***	***[e] [f] [g] [i] [j] [k] [Exp]***		***[e] [i] [Exp]***	[j] [k]	[i]		***[e] [f] [g] [Exp]***	[j] [k]	[g] [j] [k]	[g] [j] [k]	[g]			[f]
	Unlimited field of vision	[a], [b], [c], [d], [h], [l]	[b] [c] [d]			[d]	[h] [l]	[b] [c]	[d]	[b] [c] [d] [l]	[h] [l]	[a] [b] [c]				[c] [h] [l]	[a] [b] [c] [d] [h] [l]		[c] [h] [l]			[h] [l]	[l]	
	Robot can translate on the plane	***[a], [b], [c], [d], [e], [f], [g], [a], [i], [j], [k], [l], [Exp]***	***[b] [c] [d] [e] [Exp]***	[j]	[i] [k]	[d]	[f] [g] [h] [l]	***[b] [c] [e] [i] [j] [Exp]***	[d]	***[b] [c] [d] [e] [g] [i] [j] [k] [l] [Exp]***	***[e] [f] [g] [h] [i] [j] [k] [l] [Exp]***	[a] [b] [c]	***[e] [i] [Exp]***	[j] [k]	[i]	[c] [h] [l]	***[a] [b] [c] [d] [e] [f] [g] [h] [i] [Exp]***	[j] [k]	[c] [g] [h] [j] [k] [l]	[g] [j] [k]	[g]	[h] [l]	[l]	[f]
	Robot can rotate in the plane	***[a], [b], [c], [d], [e], [f], [g], [a], [i], [j], [k], [l], [Exp]***	***[b] [c] [d] [e] [Exp]***	[j]	[i] [k]	[d]	[f] [g] [h] [l]	***[b] [c] [e] [i] [j] [Exp]***	[d]	***[b] [c] [d] [e] [g] [i] [j] [k] [l] [Exp]***	***[e] [f] [g] [h] [i] [j] [k] [l] [Exp]***	[a] [b] [c]	***[e] [i] [Exp]***	[j] [k]	[i]	[c] [h] [l]	***[a] [b] [c] [d] [e] [f] [g] [h] [i] [Exp]***	[j] [k]	[c] [g] [h] [j] [k] [l]	[g] [j] [k]	[g]	[h] [l]	[l]	[f]
	Lift & Drop	[a]										[a]					[a]							
	Pull	[b], [c], [d], [g], [h], [i], [l]	[b] [c] [d]		[i]	[d]	[g] [h] [l]	[b] [c] [i]	[d]	[b] [c] [d] [g] [i] [l]	[g] [h] [i] [l]	[b] [c]	[i]		[i]	[c] [h] [l]	[b] [c] [d] [g] [h] [l]		[c] [g] [h] [l]	[g]	[g]	[h] [l]	[l]	
	Push	***[b], [c], [d], [e], [f], [g], [a], [i], [l], [j], [k], [l], [Exp]***	***[b] [c] [d] [e] [Exp]***	[j]	[i] [k]	[d]	[f] [g] [h] [l]	***[b] [c] [e] [i] [j] [Exp]***	[d]	***[b] [c] [d] [e] [g] [i] [j] [k] [l] [Exp]***	***[e] [f] [g] [h] [i] [j] [k] [l] [Exp]***	[b] [c]	***[e] [i] [Exp]***	[j] [k]	[i]	[c] [h] [l]	***[b] [c] [d] [e] [f] [g] [a] [h] [i] [Exp]***	[j] [k]	[c] [g] [h] [j] [k] [l]	[g] [j] [k]	[g]	[h] [l]	[l]	[f]
Problem class	L1	***[a], [b], [c], [e], [f], [a], [i], [j], [c], [l], [Exp]***	***[b] [c] [e] [Exp]***	[j]	[i] [k]		[f] [h] [l]	***[b] [c] [e] [i] [j] [Exp]***		***[b] [c] [e] [i] [j] [k] [l] [Exp]***	***[e] [f] [h] [i] [j] [k] [l] [Exp]***	[a] [b] [c]	***[e] [i] [Exp]***	[j] [k]	[i]	[c] [h] [l]	***[a] [b] [c] [e] [f] [h] [i] [Exp]***	[j] [k]	[c] [h] [j] [k] [l]	[j] [k]		[h] [l]	[l]	[f]
	LkM	[d], [g]	[d]			[d]	[g]		[d]	[d] [g]	[g]						[d] [g]		[g]	[g]	[g]			

CROSS COMPARISON TABLE BETWEEN HYPOTHESES AND PERFORMANCE CRITERIA			Evaluation in a simulated/real setting		Computation time	Optimality and completeness			Optimality type				Social acceptability	Number and Density of obstacles		
			Evaluation in a real-world setting	Evaluation in a simulation	Real time	Guaranteed Global Optimality	Guaranteed Local Optimality	Guaranteed Completeness	Energy optimality	Distance optimality	Time optimality	Other optimality	Mention of social norms/concerns	Maximal tested quantity of "movable obstacles" >= 20	Maximal tested quantity of "movable obstacles" < 20	Mention of the concept of obstacle density
			[c], [f], [g], [j], [k], [l]	**[a], [b], [d], [e], [h], [i], [Exp]**	**[b], [c], [d], [e], [f], [g], [i], [j], [k], [l], [Exp]**	[b], [h]	**[i], [Exp]**	[b], [c], [h]	[a], [b], [c], [e], [h], [i], [l]	**[a], [d], [f], [g], [k], [Exp]**	[h], [j], [k], [l]	**[b], [d], [f], [g], [h], [k], [Exp]**	**[b], [f], Exp**	[b], [e], [h], [i]	**[a], [c], [d], [f], [g], [j], [k], [l], Exp**	[e], [i]
Knowledge of the environment	2D metric map	**[b], [d], [e],[h], [i], [l], [Exp]**	[l]	**[b] [d] [e] [h] [i] [Exp]**	**[b] [d] [e] [i] [l] [Exp]**	[b] [h]	**[i] [Exp]**	[b] [h]	[b] [e] [h] [i] [l]	**[d] [Exp]**	[h] [l]	**[b] [d] [h] [Exp]**	**[b] [Exp]**	[b] [e] [h] [i]	**[d] [l] [Exp]**	[e] [i]
	2D costmap	[j], [k]	[j] [k]		[j] [k]					[k]	[j] [k]	[k]			[j] [k]	
	3D metric map	[a], [c], [f], [g]	[c] [f] [g]	[a]	[c] [f] [g]			[c]	[a] [c]	[a] [f] [g]		[f] [g]	[f]		[a] [c] [f] [g]	
	Complete	[a], [b], [d], [h], [l]	[l]	[a] [b] [d] [h]	[b] [d] [l]	[b] [h]		[b] [h]	[a] [b] [h] [l]	[a] [d]	[h] [l]	[b] [d] [h]	[b]	[b] [h]	[a] [d] [l]	
	Partial	**[c], [g], [Exp]**	[c] [g]	**[Exp]**	**[c] [g] [Exp]**		**[Exp]**	[c]	[c]	**[g] [Exp]**		**[g] [Exp]**	**[Exp]**		**[c] [g] [Exp]**	
	Unknown	[e], [i], [f], [j], [k]	[f] [j] [k]	[e] [i]	[e] [f] [i] [j] [k]		[i]		[e] [i]	[f] [k]	[j] [k]	[f] [k]	[f]	[e] [i]	[f] [j] [k]	[e] [i]
	Perfect data	**[a], [b], [d], [e], [i], [Exp]**		**[a] [b] [d] [e] [i] [Exp]**	**[b] [d] [e] [i] [Exp]**	[b]	**[i] [Exp]**	[b]	[a] [b] [e] [i]	**[a] [d] [Exp]**		**[b] [d] [Exp]**	**[b] [Exp]**	[b] [e] [i]	**[a] [d] [Exp]**	[e] [i]
	Approximative data	[c], [f], [g], [h], [j], [k], [l]	[c] [f] [g] [j] [k] [l]	[h]	[c] [f] [g] [j] [k] [l]	[h]		[c] [h]	[c] [h] [l]	[f] [g] [k]	[h] [j] [k] [l]	[f] [g] [h] [k]	[f]	[h]	[c] [f] [g] [j] [k] [l]	
	Free unknown space hypothesis	**[e], [f], [i], [j], [k], [Exp]**	[f] [j] [k]	**[e] [i] [Exp]**	**[e] [f] [i] [j] [k] [Exp]**		**[i] [Exp]**		[e] [i]	**[f] [k] [Exp]**	[j] [k]	**[f] [k] [Exp]**	**[f] [Exp]**	[e] [i]	**[f] [j] [k] [Exp]**	[e] [i]
Obstacle characteristics	Naive 2D projection	[a], [b], [e], [g], [h], [i], [l]	[g] [l]	[a] [b] [e] [h] [i]	[b] [e] [g] [i] [l]	[b] [h]	[i]	[b] [h]	[a] [b] [e] [h] [i] [l]	[a] [g]	[h] [l]	[b] [g] [h]	[b]	[b] [e] [h] [i]	[a] [g] [l]	[e] [i]
	2D Projection using Convex-Hull	**[c], [d], [f], [Exp]**	[c] [f]	**[d] [Exp]**	**[c] [d] [f] [Exp]**		**[Exp]**	[c]	[c]	**[d] [f] [Exp]**		**[d] [f] [Exp]**	**[f] [Exp]**		**[c] [d] [f] [Exp]**	
	Any obstacle types	[f], [g], [h], [i], [j], [k], [l]	[f] [g] [j] [k] [l]	[h] [i]	[f] [g] [i] [j] [k] [l]	[h]	[i]	[h]	[h] [i] [l]	[f] [g] [k]	[h] [j] [k] [l]	[f] [g] [h] [k]	[f]	[h] [i]	[f] [g] [j] [k] [l]	[i]
	Only polygonal obstacles	**[a], [b], [c], [d], [Exp]**	[c]	**[a] [b] [d] [Exp]**	**[b] [c] [d] [Exp]**	[b]	**[Exp]**	[b] [c]	[a] [b] [c]	**[a] [d] [Exp]**		**[b] [d] [Exp]**	**[b] [Exp]**	[b]	**[a] [c] [d] [Exp]**	
	Only rectangular obstacles	[e]		[e]	[e]				[e]					[e]		[e]
	Human obstacle															
	Moving obstacle	**[k], [Exp]**	[k]	**[Exp]**	**[k] [Exp]**		**[Exp]**			**[k] [Exp]**	[k]	**[k] [Exp]**	**[Exp]**		**[k] [Exp]**	
	Metadata on obstacle's physics	[a], [b], [c], [d], [h]	[c]	[a] [b] [d] [h]	[b] [c] [d]	[b] [h]		[b] [c] [h]	[a] [b] [c] [h]	[a] [d]	[h]	[b] [d] [h]	[b]	[b] [h]	[a] [c] [d]	
	Obstacle can be translated in 2D plane	**[a], [b], [c], [d], [e], [f], [g], [h], [i], [j], [k], [l], [Exp]**	[c] [f] [g] [j] [k] [l]	**[a] [b] [d] [e] [h] [i] [Exp]**	**[b] [c] [d] [e] [f] [g] [i] [j] [k] [l] [Exp]**	[b] [h]	**[i] [Exp]**	[b] [c] [h]	[a] [b] [c] [e] [h] [i] [l]	**[a] [d] [f] [g] [k] [Exp]**	[h] [j] [k] [l]	**[b] [d] [f] [g] [h] [k] [Exp]**	**[b] [f] [Exp]**	[b] [e] [h] [i]	**[a] [c] [d] [f] [g] [j] [k] [l] [Exp]**	[e] [i]
	Translation limited to the 2D plane axes	[e]		[e]	[e]				[e]					[e]		[e]
	Obstacle can be rotated in the normal to the 2D plane	[a], [b], [d], [g], [h], [l]	[g] [l]	[a] [b] [d] [h]	[b] [d] [g] [l]	[b] [h]		[b] [h]	[a] [b] [h] [l]	[a] [d] [g]	[h] [l]	[b] [d] [g] [h]	[b]	[b] [h]	[a] [d] [g] [l]	
Robot characteristics	HRP2 Robot	[a], [c], [f]	[c] [f]	[a]	[c] [f]			[c]	[a] [c]	[a] [f]		[f]	[f]		[a] [c] [f]	
	PR2 Robot	[g]	[g]		[g]					[g]		[g]			[g]	
	GOLEM Krang Robot	[h], [l]	[l]	[h]	[l]	[h]		[h]	[h] [l]		[h] [l]	[h]		[h]	[l]	
	Custom robot vehicle for MAGIC 2010 Competition	[j], [k]	[j] [k]		[j] [k]					[k]	[j] [k]	[k]			[j] [k]	
	Pepper Robot	**[Exp]**		**[Exp]**	**[Exp]**		**[Exp]**			**[Exp]**		**[Exp]**	**[Exp]**		**[Exp]**	
	Nondescript humanoid robot	[b], [d]		[b] [d]	[b] [d]	[b]		[b]	[b]	[d]		[b] [d]	[b]	[b]	[d]	
	Nondescript wheeled robot	[e], [i]		[e] [i]	[e] [i]		[i]		[e] [i]					[e] [i]		[e] [i]
	Limited field of vision	**[e], [f], [g], [i], [j], [k], [Exp]**	[f] [g] [j] [k]	**[e] [i] [Exp]**	**[e] [f] [g] [i] [j] [k] [Exp]**		**[i] [Exp]**		[e] [i]	**[f] [g] [k] [Exp]**	[j] [k]	**[f] [g] [k] [Exp]**	**[f] [Exp]**	[e] [i]	**[f] [g] [j] [k] [Exp]**	[e] [i]
	Unlimited field of vision	[a], [b], [c], [d], [h], [l]	[c] [l]	[a] [b] [d] [h]	[b] [c] [d] [l]	[b] [h]		[b] [c] [h]	[a] [b] [c] [h] [l]	[a] [d]	[h] [l]	[b] [d] [h]	[b]	[b] [h]	[a] [c] [d] [l]	
	Robot can translate on the plane	**[a], [b], [c], [d], [e], [f], [g], [h], [i], [j], [k], [l], [Exp]**	[c] [f] [g] [j] [k] [l]	**[a] [b] [d] [e] [h] [i] [Exp]**	**[b] [c] [d] [e] [f] [g] [i] [j] [k] [l] [Exp]**	[b] [h]	**[i] [Exp]**	[b] [c] [h]	[a] [b] [c] [e] [h] [i] [l]	**[a] [d] [f] [g] [k] [Exp]**	[h] [j] [k] [l]	**[b] [d] [f] [g] [h] [k] [Exp]**	**[b] [f] [Exp]**	[b] [e] [h] [i]	**[a] [c] [d] [f] [g] [j] [k] [l] [Exp]**	[e] [i]
	Robot can rotate in the plane	**[a], [b], [c], [d], [e], [f], [g], [h], [i], [j], [k], [l], [Exp]**	[c] [f] [g] [j] [k] [l]	**[a] [b] [d] [e] [h] [i] [Exp]**	**[b] [c] [d] [e] [f] [g] [i] [j] [k] [l] [Exp]**	[b] [h]	**[i] [Exp]**	[b] [c] [h]	[a] [b] [c] [e] [h] [i] [l]	**[a] [d] [f] [g] [k] [Exp]**	[h] [j] [k] [l]	**[b] [d] [f] [g] [h] [k] [Exp]**	**[b] [f] [Exp]**	[b] [e] [h] [i]	**[a] [c] [d] [f] [g] [j] [k] [l] [Exp]**	[e] [i]
	Lift & Drop	[a]		[a]					[a]	[a]					[a]	
	Pull	[b], [c], [d], [g], [h], [i], [l]	[c] [g] [l]	[b] [d] [h] [i]	[b] [c] [d] [g] [i] [l]	[b] [h]	[i]	[b] [c] [h]	[b] [c] [h] [i] [l]	[d] [g]	[h] [l]	[b] [d] [g] [h]	[b]	[b] [h] [i]	[c] [d] [g] [l]	[i]
	Push	**[b], [c], [d], [e], [f], [g], [h], [i], [j], [k], [l], [Exp]**	[c] [f] [g] [j] [k] [l]	**[b] [d] [e] [h] [i] [Exp]**	**[b] [c] [d] [e] [f] [g] [i] [j] [k] [l] [Exp]**	[b] [h]	**[i] [Exp]**	[b] [c] [h]	[b] [c] [e] [h] [i] [l]	**[d] [f] [g] [k] [Exp]**	[h] [j] [k] [l]	**[b] [d] [f] [g] [h] [k] [Exp]**	**[b] [f] [Exp]**	[b] [e] [h] [i]	**[c] [d] [f] [g] [j] [k] [l] [Exp]**	[e] [i]
Problem class	L1	**[a], [b], [c], [e], [f], [h], [i], [j], [k], [l], [Exp]**	[c] [f] [j] [k] [l]	**[a] [b] [e] [h] [i] [Exp]**	**[b] [c] [e] [f] [i] [j] [k] [l] [Exp]**	[b] [h]	**[i] [Exp]**	[b] [c] [h]	[a] [b] [c] [e] [h] [i] [l]	**[a] [f] [k] [Exp]**	[h] [j] [k] [l]	**[b] [f] [h] [k] [Exp]**	**[b] [f] [Exp]**	[b] [e] [h] [i]	**[a] [c] [f] [j] [k] [l] [Exp]**	[e] [i]
	LkM	[d], [g]	[g]	[d]	[d] [g]					[d] [g]		[d] [g]			[d] [g]	

CROSS COMPARISON TABLE BETWEEN PERFORMANCE CRITERIA AND APPROACHES

			Path Planning Algorithm(s) and heuristics								Evaluation and evolution of an obstacle's "movable" characteristic and its associated cost					Object manipulation maneuver planning			Planning taking uncertainty into account					
			A*	ARA*	D* Lite	BFS	RRT	Standard Heuristic for Path Planning	Custom Heuristic for Path Planning	Supplementary Heuristics	"Movability" (re)evaluated on runtime	Manipulation cost depends on the obstacle's physics metadata	Manipulation cost depends on a constant common to all obstacles	Cost is estimated on runtime	Cost is pre-estimated by a heuristic	Kinematic/Friction constraints taken into account	Limited grasping points number	No concern about grasping points	Adaptive obstacle approach procedures	Use of a Kalman filter	Use of e-shadows	Use of PRM + MDP + MonteCarlo	Use of PBRL	Pointcloud correction
			[b], [c], [d], [e], [Exp]	[j]	[i], [k]	[d]	[f], [g], [h], [l]	[b], [c], [e], [i], [j], [Exp]	[d]	[b], [c], [d], [e], [g], [i], [j], [l], [Exp]	[e], [f], [g], [h], [i], [j], [k], [l], [Exp]	[a], [b]. [c]	[e], [i], [Exp]	[j], [k]	[i]	[c], [h], [l]	[a], [b], [c], [d], [e], [f], [g], [i], [l], [Exp]	[j], [k]	[c], [g], [h], [j], [k], [l]	[g], [j], [k]	[g]	[h], [l]	[l]	[f]
Evaluation in a simulated/real setting	Evaluation in a real-world setting	[c], [f], [g], [j], [k], [l]	[c]	[j]	[k]		[f] [g] [l]	[c] [j]		[c] [g] [j] [k] [l]	[f] [g] [j] [k] [l]	[c]		[j] [k]		[c] [l]	[c] [f] [g] [l]	[j] [k]	[c] [g] [j] [k] [l]	[g] [j] [k]	[g]	[l]	[l]	[f]
	Evaluation in a simulation	[a], [b], [d], [e], [h], [i], [Exp]	[b] [d] [e] [Exp]		[i]	[d]	[h]	[b] [e] [i] [Exp]	[d]	[b] [d] [e] [i] [Exp]	[e] [h] [i] [Exp]	[a] [b]	[e] [i] [Exp]		[i]	[h]	[a] [b] [d] [e] [h] [Exp]		[h]			[h]		
Computation time	Real time	[b], [c], [d], [e], [f], [g], [i], [j], [k], [l], [Exp]	[b] [c] [d] [e] [Exp]	[j]	[i] [k]	[d]	[f] [g] [l]	[b] [c] [e] [i] [j] [Exp]	[d]	[b] [c] [d] [e] [g] [i] [j] [Exp]	[e] [f] [g] [i] [j] [k] [l] [Exp]	[b] [c]	[e] [i] [Exp]	[j] [k]	[i]	[c] [l]	[b] [c] [d] [e] [f] [g] [i] [Exp]	[j] [k]	[c] [g] [j] [k] [l]	[g] [j] [k]	[g]	[l]	[l]	[f]
Optimality and completeness	Guaranteed Global Optimality	[b], [h]	[b]				[h]	[b]		[b]	[h]	[b]				[h]	[b] [h]		[h]			[h]		
	Guaranteed Local Optimality	[i], [Exp]	[Exp]		[i]			[i] [Exp]		[i] [Exp]	[i] [Exp]		[i] [Exp]		[i]		[Exp]							
	Guaranteed Completeness	[b], [c], [h]	[b] [c]				[h]	[b] [c]		[b] [c]	[h]	[b] [c]				[c] [h]	[b] [c] [h]		[c] [h]			[h]		
Optimality type	Energy optimality	[a], [b], [c], [e], [h], [i], [l]	[b] [c] [e]		[i]		[h] [l]	[b] [c] [e] [i]		[b] [c] [e] [i] [l]	[e] [h] [i] [l]	[a] [b] [c]	[e] [i]		[i]	[c] [h] [l]	[a] [b] [c] [e] [h] [l]		[c] [h] [l]			[h] [l]	[l]	
	Distance optimality	[a], [d], [f], [g], [k], [Exp]	[d] [Exp]		[k]	[d]	[f] [g]	[Exp]	[d]	[d] [g] [k] [Exp]	[f] [g] [k] [Exp]	[a]	[Exp]	[k]			[a] [d] [f] [g] [Exp]	[k]	[g] [k]	[g] [k]	[g]			[f]
	Time optimality	[h], [j], [k], [l]		[j]	[k]		[h] [l]	[j]		[j] [k] [l]	[h] [j] [k] [l]			[j] [k]		[h] [l]	[h] [l]	[j] [k]	[h] [j] [k] [l]	[j] [k]		[h] [l]	[l]	
	Other optimality	[b], [d], [f], [g], [h], [k], [Exp]	[b] [d] [Exp]		[k]	[d]	[f] [g] [h]	[b] [Exp]	[d]	[b] [d] [g] [k] [Exp]	[f] [g] [h] [k] [Exp]	[b]	[Exp]	[k]		[h]	[b] [d] [f] [g] [h] [Exp]	[k]	[g] [h] [k]	[g] [k]	[g]	[h]		[f]
Social acceptability	Mention of social norms/concerns	[b], [f], [Exp]	[b] [Exp]				[f]	[b] [Exp]		[b] [Exp]	[f] [Exp]	[b]	[Exp]				[b] [f] [Exp]							[f]
Number and Density of obstacles	Maximal tested quantity of "movable obstacles" >= 20	[b], [e], [h], [i]	[b] [e]		[i]		[h]	[b] [e] [i]		[b] [e] [i]	[e] [h] [i]	[b]	[e] [i]		[i]	[h]	[b] [e] [h]		[h]			[h]		
	Maximal tested quantity of "movable obstacles" < 20	[a], [c], [d], [f], [g], [j], [k], [l], [Exp]	[c] [d] [Exp]	[j]	[k]	[d]	[f] [g] [l]	[c] [j] [Exp]	[d]	[c] [d] [g] [j] [k] [l] [Exp]	[d] [g] [j] [k] [l] [Exp]	[a] [c]	[Exp]	[j] [k]		[c] [l]	[c] [d] [f] [g] [i] [Exp]	[j] [k]	[c] [g] [j] [k] [l]	[g] [j] [k]	[g]	[l]	[l]	[f]
	Mention of the concept of obstacle density	[e], [i]	[e]		[i]			[e] [i]		[e] [i]	[e] [i]		[e] [i]		[i]		[e]							

Bibliography

[1] C. Amato et al. “Planning for decentralized control of multiple robots under uncertainty”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015 IEEE International Conference on Robotics and Automation (ICRA). May 2015, pp. 1241–1248. DOI: 10.1109/ICRA.2015.7139350.

[2] Ohad Ben-Shahar and E. Rivlin. “Practical pushing planning for rearrangement tasks”. In: *IEEE Transactions on Robotics and Automation* 14.4 (Aug. 1998), pp. 549–565. ISSN: 1042-296X. DOI: 10.1109/70.704220.

[3] Jur van den Berg et al. “Path Planning among Movable Obstacles: A Probabilistically Complete Approach”. In: *Algorithmic Foundation of Robotics VIII*. Springer Tracts in Advanced Robotics. Springer, Berlin, Heidelberg, 2009, pp. 599–614. ISBN: 978-3-642-00311-0 978-3-642-00312-7. DOI: 10.1007/978-3-642-00312-7_37. URL: https://link.springer.com/chapter/10.1007/978-3-642-00312-7_37 (visited on 02/28/2018).

[4] F. Bonsignorio and A. P. del Pobil. “Toward Replicable and Measurable Robotics Research [From the Guest Editors]”. In: *IEEE Robotics Automation Magazine* 22.3 (Sept. 2015), pp. 32–35. ISSN: 1070-9932. DOI: 10.1109/MRA.2015.2452073.

[5] P. C. Chen and Y. K. Hwang. “Practical path planning among movable obstacles”. In: *1991 IEEE International Conference on Robotics and Automation Proceedings*. 1991 IEEE International Conference on Robotics and Automation Proceedings. Apr. 1991, 444–449 vol.1. DOI: 10.1109/ROBOT.1991.131618.

[6] 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 IEEE International Conference on Robotics and Automation (ICRA). May 2014, pp. 2771–2778. DOI: 10.1109/ICRA.2014.6907256.

[7] 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 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Sept. 2015, pp. 6121–6128. DOI: 10.1109/IROS.2015.7354249.

[8] Chiara Fulgenzi. “Autonomous navigation in dynamic uncertain environment using probabilistic models of perception and collision risk prediction.” PhD thesis. Institut National Polytechnique de Grenoble - INPG, June 8, 2009. URL: https://tel.archives-ouvertes.fr/tel-00398055/document (visited on 03/29/2018).

[9] E. Guglielmelli. “Research Reproducibility and Performance Evaluation for Dependable Robots [From the Editor’s Desk]”. In: *IEEE Robotics Automation Magazine* 22.3 (Sept. 2015), pp. 4–4. ISSN: 1070-9932. DOI: 10.1109/MRA.2015.2470815.

[10] Fabrice Jumel, Jacques Saraydaryan, and Olivier Simonin. “Mapping likelihood of encountering humans: application to path planning in crowded environment”. In: *The European Conference on Mobile Robotics (ECMR)*. Proceedings of ECMR 2017. Paris, France, Sept. 2017. URL: https://hal.archives-ouvertes.fr/hal-01588815 (visited on 03/05/2018).

[11] 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 IEEE International Conference on Robotics and Automation. May 2011, pp. 1470–1477. DOI: 10.1109/ICRA.2011.5980391.

[12] Y. Kakiuchi et al. "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 IEEE/RSJ International Conference on Intelligent Robots and Systems. Oct. 2010, pp. 1696–1701. DOI: 10.1109/IROS.2010.5650206.

[13] S. Koenig and M. Likhachev. "Fast replanning for navigation in unknown terrain". In: *IEEE Transactions on Robotics* 21.3 (June 2005), pp. 354–363. ISSN: 1552-3098. DOI: 10.1109/TRO.2004.838026.

[14] Thibault Kruse et al. "Human-aware robot navigation: A survey". In: *Robotics and Autonomous Systems* 61.12 (Dec. 1, 2013), pp. 1726–1743. ISSN: 0921-8890. DOI: 10.1016/j.robot.2013.05.007. URL: http://www.sciencedirect.com/science/article/pii/S0921889013001048 (visited on 03/05/2018).

[15] 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 IEEE International Conference on Robotics and Automation. May 2013, pp. 3832–3838. DOI: 10.1109/ICRA.2013.6631116.

[16] 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 IEEE-RAS International Conference on Humanoid Robots. Nov. 2014, pp. 86–91. DOI: 10.1109/HUMANOIDS.2014.7041342.

[17] M. Levihn et al. "Autonomous environment manipulation to assist humanoid locomotion". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014 IEEE International Conference on Robotics and Automation (ICRA). May 2014, pp. 4633–4638. DOI: 10.1109/ICRA.2014.6907536.

[18] M. Levihn et al. "Foresight and reconsideration in hierarchical planning and execution". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. Nov. 2013, pp. 224–231. DOI: 10.1109/IROS.2013.6696357.

[19] Martin Levihn, Jonathan Scholz, and Mike Stilman. "Hierarchical Decision Theoretic Planning for Navigation Among Movable Obstacles". In: *Algorithmic Foundations of Robotics X*. Springer Tracts in Advanced Robotics. Springer, Berlin, Heidelberg, 2013, pp. 19–35. ISBN: 978-3-642-36278-1 978-3-642-36279-8. DOI: 10.1007/978-3-642-36279-8_2. URL: https://link.springer.com/chapter/10.1007/978-3-642-36279-8_2 (visited on 02/28/2018).

[20] Martin Levihn and Mike Stilman. *Efficient Opening Detection*. Technical Report. Georgia Institute of Technology, 2011. URL: https://smartech.gatech.edu/handle/1853/40954 (visited on 03/25/2018).

[21] Dennis Nieuwenhuisen, A. Frank van der Stappen, and Mark H. Overmars. "An Effective Framework for Path Planning Amidst Movable Obstacles". In: *Algorithmic Foundation of Robotics VII*. Springer Tracts in Advanced Robotics. Springer, Berlin, Heidelberg, 2008, pp. 87–102. ISBN: 978-3-540-68404-6 978-3-540-68405-3. DOI: 10.1007/978-3-540-68405-3_6. URL: https://link.springer.com/chapter/10.1007/978-3-540-68405-3_6 (visited on 02/28/2018).

[22] K. Okada et al. "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 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566). Vol. 2. Sept. 2004, 1174–1179 vol.2. DOI: 10.1109/IROS.2004.1389555.

[23] K. Okada et al. "Humanoid motion generation system on HRP2-JSK for daily life environment". In: *IEEE International Conference Mechatronics and Automation, 2005*. IEEE International Conference Mechatronics and Automation, 2005. Vol. 4. July 2005, 1772–1777 Vol. 4. DOI: 10.1109/ICMA.2005.1626828.

[24] 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 IEEE/RSJ International Conference on Intelligent Robots and Systems. Sept. 2014, pp. 2627–2632. DOI: 10.1109/IROS.2014.6942921.

[25] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *arXiv:1612.08242 [cs]* (Dec. 25, 2016). arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242 (visited on 08/22/2018).

[26] J. Rios-Martinez, A. Spalanzani, and C. Laugier. "From Proxemics Theory to Socially-Aware Navigation: A Survey". In: *International Journal of Social Robotics* 7.2 (Apr. 1, 2015), pp. 137–153. ISSN: 1875-4791, 1875-4805. DOI: 10.1007/s12369-014-0251-1. URL: https://link.springer.com/article/10.1007/s12369-014-0251-1 (visited on 03/05/2018).

[27] 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 IEEE/RSJ International Conference on Intelligent Robots and Systems. Sept. 2011, pp. 2014–2019. DOI: 10.1109/IROS.2011.6094496.

[28] Jorge Rios-Martinez. "Socially-Aware Robot Navigation : combining Risk Assessment and Social Conventions". In: (Jan. 8, 2013).

[29] A. Sahbani, S. El-Khoury, and P. Bidaud. "An overview of 3D object grasp synthesis algorithms". In: *Robotics and Autonomous Systems*. Autonomous Grasping 60.3 (Mar. 1, 2012), pp. 326–336. ISSN: 0921-8890. DOI: 10.1016/j.robot.2011.07.016. URL: http://www.sciencedirect.com/science/article/pii/S0921889011001485 (visited on 08/20/2018).

[30] J. Scholz et al. "Navigation Among Movable Obstacles with learned dynamic constraints". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Oct. 2016, pp. 3706–3713. DOI: 10.1109/IROS.2016.7759546.

[31] Mike Stilman. "Navigation Among Movable Obstacles". PhD thesis. Pittsburgh, PA: Carnegie Mellon University, Oct. 2007. 145 pp. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.591.3574&rep=rep1&type=pdf.

[32] Mike Stilman and James Kuffner. "Planning Among Movable Obstacles with Artificial Constraints". In: *The International Journal of Robotics Research* 27.11 (Nov. 1, 2008), pp. 1295–1307. ISSN: 0278-3649. DOI: 10.1177/0278364908098457. URL: https://doi.org/10.1177/0278364908098457 (visited on 02/28/2018).

[33] Mike Stilman and James J. Kuffner. "Navigation among movable obstacles: real-time reasoning in complex environments". In: *International Journal of Humanoid Robotics* 02.4 (Dec. 1, 2005), pp. 479–503. ISSN: 0219-8436. DOI: 10.1142/S0219843605000545. URL: http://www.worldscientific.com/doi/abs/10.1142/S0219843605000545 (visited on 02/28/2018).

[34] Mike Stilman et al. “Planning and executing navigation among movable obstacles”. In: *Advanced Robotics* 21.14 (Jan. 1, 2007), pp. 1617–1634. ISSN: 0169-1864. DOI: 10.1163/156855307782227408. URL: https://doi.org/10.1163/156855307782227408 (visited on 02/28/2018).

[35] P. Trautman and A. Krause. “Unfreezing the robot: Navigation in dense, interacting crowds”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems. Oct. 2010, pp. 797–803. DOI: 10.1109/IROS.2010.5654369.

[36] Ruffin White and Henrik Christensen. “ROS and Docker”. In: *Robot Operating System (ROS): The Complete Reference (Volume 2)*. Ed. by Anis Koubaa. Studies in Computational Intelligence. Cham: Springer International Publishing, 2017, pp. 285–307. ISBN: 978-3-319-54927-9. DOI: 10.1007/978-3-319-54927-9_9. URL: https://doi.org/10.1007/978-3-319-54927-9_9 (visited on 08/20/2018).

[37] Gordon Wilfong. “Motion planning in the presence of movable obstacles”. In: *Annals of Mathematics and Artificial Intelligence* 3.1 (Mar. 1, 1991), pp. 131–150. ISSN: 1012-2443, 1573-7470. DOI: 10.1007/BF01530890. URL: https://link.springer.com/article/10.1007/BF01530890 (visited on 02/28/2018).

[38] Hai-Ning 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 IEEE/RSJ International Conference on Intelligent Robots and Systems. Oct. 2010, pp. 1433–1438. DOI: 10.1109/IROS.2010.5649744.