NATIONAL INSTITUTE OF APPLIED SCIENCES OF LYON

RESEARCH REPORT

First steps toward integrating social and dynamic concerns in the domain of Navigation Among Movable Obstacles

---

Author:
Benoit RENAULT

Supervisors:
Dr. Fabrice JUMEL
Dr. Jacques SARAYDARYAN
Dr. Olivier SIMONIN

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

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

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

August 16, 2018

Résumé: L'objectif est de permettre la navigation optimale dun seul robot dans un environnement intérieur partiellement connu, dynamique et modifiable, en respectant les conventions sociales.

Mots-clefs: NAMO, Navigation en milieu modifiable, Navigation Sociale

Abstract: The objective is to allow optimal navigation among movable obstacles of a single robot in a partially known interior setting, while respecting social conventions.

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

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 perfectly fine in a static context (nothing moves, at the exception of the robot). However, from the description of a human environment given above, this is not sufficient.

On one hand, planning in dynamic environments populated with humans is its very own domain [11, 22]. The INRIA Chroma team (to which the author is affiliated) is active in this domain (among others), and proposed algorithms that optimize the generation of trajectories by managing the risk of colliding with obstacles [6, 24], respecting social conventions such as avoiding human interaction spaces [21, 23], or predicting the trajectories of moving obstacles [7], . . .

On the other hand, planning in alterable environments is also its own research field called NAMO (Navigation Among Movable Obstacles)[26]. Indeed, the problem of generating a navigation plan that may imply the manipulation of obstacles is very vast, and has been dealt with in a great variety of contexts, detailed in Chapter 2.

To this day, from our research, it seems that these two domains 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

Therefore, the overall objectives of this work are to:

- explore the NAMO domain and extract characteristics that allow to sort through existing works,
- identify both new concerns and bridges between this domain and the one of dynamic and social navigation,
- and finally build upon existing work to propose solutions to the previously identified concerns.

As a context for formulating, comparing and evaluating hypotheses, we will assume that our goal implementation platform is the Pepper robot [1] since it is the standard platform for the Robocup@Home[2]. This international-level competition provides service robotics challenges to evaluate solutions proposed by researchers and students, and since the INRIA Chroma team participates to it, it makes for a good goal setting. Furthermore, the team uses ROS [3] to command the robot, further clarifying our goal context. 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,

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 B and A gather comparison tables and pseudocode representations of algorithm propositions.

[1] http://doc.aldebaran.com/2-4/family/pepper_technical/index_dev_pepper.html
[2] http://www.robocupathome.org/
[3] ROS, the Robot Operating System: http://www.ros.org/

Chapter 2

Navigation Among Movable Obstacles: state of the art

2.1 Determining appropriate comparison criteria

When exploring an unknown research domain, it is appropriate to establish comparison criteria that allow to understand how solutions differ from one another. We divide these criteria 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

This first category exposes what is the necessary context for a proposition.

- **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 (is it free or 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 characteristic like center of gravity, weight, moment of inertia, ...), autonomous movement or not of the considered obstacles all condition the final algorithm proposition.

- **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 [28] 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 [1]. 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. The notation is further refined in another paper [27] 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

This second category explains what a proposition is in itself.

- **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 obstacles "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 off 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 actuating capabilities, and therefore, uncertainty about the state of the 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

This last category regroups means to evaluate the relevance of a proposition.

- **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. Bringing an algorithm that has only, until now, been used in simulation to reality would be valuable enough to justify a publication.

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

- **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). 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 type** 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 effiency 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 recapitulates the above-mentioned criteria.

Hypotheses
Previous knowledge of the map
Obstacle characteristics
Robot characteristics
Problem type

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

Performance criteria
Evaluation in a simulated/real setting
Computation time
Local/global Optimality
Optimality in distance/time/energy...
Social acceptability
Number and Density of obstacles

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. Emphasized and bold cells with the "[Exp]" reference are criteria that are validated by our proposition. 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 correspondence between these letters and articles can be found in the detailed Comparison tables Appendix, but also in the table below that 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	[19]	[28]	[29]	[27]	[31]	[9]	[15]	[12]	[13]	[4]	[5]	[25]

Knowledge of the environment								
2D metric map	2D costmap	3D metric map	Complete	Partial	Unknown	Perfect data	Approximative data	Free unknown space hypothesis
[b], [d], [e],[h], [i], [l], [Exp]	[j], [k]	[a], [c], [f], [g]	[a], [b], [d], [h], [l]	***[c], [g], [Exp]***	[e], [i], [f], [j], [k]	***[a], [b], [d], [e], [i], [Exp]***	[c], [f], [g], [h], [j], [k], [l]	***[e], [f], [i], [j], [k], [Exp]***

Obstacle characteristics										
Naive 2D projection	2D Projection using Convex-Hull	Any obstacle types	Only polygonal obstacles	Only rectangular obstacles	Human obstacle	Moving obstacle	Metadata on obstacle's physics	Obstacle can be translated in 2D plane	Translation limited to the 2D plane axes	Obstacle can be rotated in the normal to the 2D plane
[a], [b], [e], [g], [h], [i], [l]	***[c], [d], [f], [Exp]***	[f], [g], [h], [i], [j], [k], [l]	***[a], [b], [c], [d], [Exp]***	[e]		***[k], [Exp]***	[a], [b], [c], [d], [h]	***[a], [b], [c], [d], [e], [f], [g], [h], [i], [j], [k], [l], [Exp]***	[e]	[a], [b], [d], [g], [h], [l]

Robot characteristics						
HRP2 Robot	PR2 Robot	GOLEM Krang Robot	Custom robot vehicle for MAGIC 2010 Competition	Pepper Robot	Nondescript humanoid robot	Nondescript wheeled robot
[a], [c], [f]	[g]	[h], [l]	[j], [k]	***[Exp]***	[b], [d]	[e], [i]

Robot characteristics							Problem class	
Limited field of vision	Unlimited field of vision	Robot can translate on the plane	Robot can rotate in the plane	Lift & Drop	Pull	Push	L1	LkM
[e], [f], [g], [i], [j], [k], [Exp]	[a], [b], [c], [d], [h], [l]	***[a], [b], [c], [d], [e], [f], [g], [h], [i], [j], [k], [l], [Exp]***	***[a], [b], [c], [d], [e], [f], [g], [h], [i], [j], [k], [l], [Exp]***	[a]	[b], [c], [d], [g], [h], [i], [l]	***[b], [c], [d], [e], [f], [g], [h], [i], [j], [k], [l], [Exp]***	***[a], [b], [c], [e], [f], [h], [i], [j], [k], [l], [Exp]***	[d], [g]

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], [l], [Exp]	[b], [h]	[i], [Exp]	[b], [c], [h]

FIGURE 2.4: Performance criteria comparison tables

2.2.2 Conclusions

The first conclusion we can draw from our research 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, Figure 2.2 shows 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 [28]. As many of the papers [28, 29, 27, 31, 12, 13, 25] 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 [30]. 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" [27]. This is why the robot are often limited to push or pull actions, following a translation movement in a single direction in the propositions. When they are not so, like in [19], 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 interaction with the environment, and with that, the chance that something might go wrong: especially, as is mentioned in [29], the robot might lose its balance.

Figure 2.2 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 [4], although the tested solution is arguably not capable of handling problems as complex as the propositions of Stilman and Levihn.

Figure 2.3 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 [5], since it allows him to potentially manage autonomously moving obstacles, since it is based off the D*Lite algorithm (see [10] and [6]), but no experiment with a changing environment is shown. Almost all papers use some sort of heuristic, be it for the path finding subroutine or choosing which obstacle to evaluate, and some of these heuristic come at the cost of optimality [28, 31]. Some papers [29, 15, 12, 4, 5, 25] offer means to take uncertainty into account. All of them have adaptive approach procedures that are executed when nearing the obstacle, but some use much more elaborate approaches to this end, like using probabilistic models [12, 25].

It is noteworthy that many of the algorithms do not focus on guaranteeing the optimality or even the completeness of the plans they produce. This can easily be explained by the fact that, in robotics, real-time performance is paramount. However, sensible approaches like [28, 31], where the proposition contains an original algorithm that is optimal, but also a modified version that improves performance at the cost of optimality, are more interesting, as it is often easier to improve the computational performance of an algorithm by sacrificing its optimality, than trying to make a non-optimal algorithm optimal.

In the selected papers, about half make propositions to deal with an unkwown or partially known environment (which is definitely correlated with the fact that the robot is considered to

[2] https://en.wikipedia.org/wiki/Sokoban

[3] Characteritics: 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

have a limited field of visions), 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 related to NAMO [3, 20, 18, 2, 8, 16, 14] 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. [9] that brought the first extension to NAMO in unknown environments [13] 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). Wu and Levihn were the ones to formulate a locally optimal algorithm for NAMO in completely unknown environments [31, 13]. 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 [15]. Clingerman also proposed later another solution for NAMO in unknown environments [4, 5], however this solution does not consider the manipulation of an obstacle as an action in itself and simply make 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. Also, if Stilman [28] and Kakiuchi [9] 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 NAMO algorithm to social conventions or norms.

2.2.3 Situating our work in the established context

In the end, our choice of algorithm goes to the solution proposed by Wu and improved by Levihn [31, 13], 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 bothersome since we did not want to spend time on grasping problematics anyway.

Also, it is noteworthy that the entire proposition was actually formulated with pseudocode, making it easier to implement. 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.

Chapter 3

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

Given the state of the art presented in the previous chapter and what we aimed to achieve in the available time, we chose to build upon 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 that 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). This is best shown in figure 3.1.

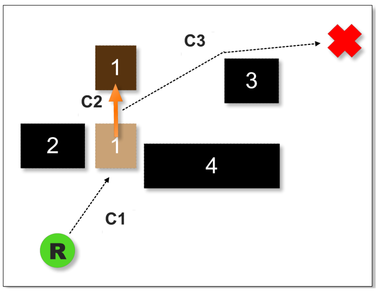


FIGURE 3.1: Figure describing the three plan components when pushing an object, as published in [31].

In their first article [31], 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. In their second article [13], several other improvements on performance are proposed that restore the local optimality of the proposition. By locally optimal, we mean that the algorithm will **always** choose the **best** plan given its current, limited knowledge of the environment.

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.

Algorithm 1 BASELINE(R_{Init}, R_{Goal})

1: $R \Leftarrow R_{Init}$;
2: $\mathcal{O} \Leftarrow \emptyset$; {set of objects}
3: $p_{opt} \Leftarrow A^*(R_{Init}, R_{Goal})$;
4: **while** $R \neq R_{Goal}$ **do**
5:   $\mathcal{O}_{new} \Leftarrow$GET-NEW-INFORMATION();
6:   **if** $\mathcal{O}_{new} \neq \emptyset$ **then**
7:     $\mathcal{O} = \mathcal{O} \cup \mathcal{O}_{new}$;
8:     **for** each $o \in \mathcal{O}$ **do**
9:       **for** each possible push direction d on o **do**
10:         $p \Leftarrow$ EVALUATE-ACTION(o,d);
11:         **if** $p.cost < p_{opt}.cost$ **then**
12:           $p_{opt} = p$;
13:         **end if**
14:       **end for**
15:     **end for**
16:   **end if**
17:   $R \Leftarrow$ Next step in p_{opt};
18: **end while**

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

Algorithm 2 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

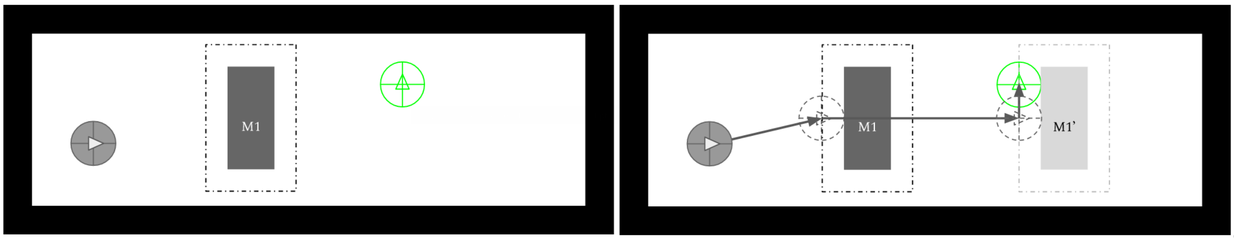
FIGURE 3.2: Baseline Algorithm as published by Wu et. al. in [31]

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 3.5a, 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 could open new, better routes**): given that we can assume that our current optimal plan was optimal before the discovery of the new obstacle, we need only reconsider it if the new obstacle actually prevents it from coming to fruition.

Second Optimization Stop simulating pushes in a given direction before it becomes costlier than the current valid optimal plan, thanks to a bound (Figure 3.5b, 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.

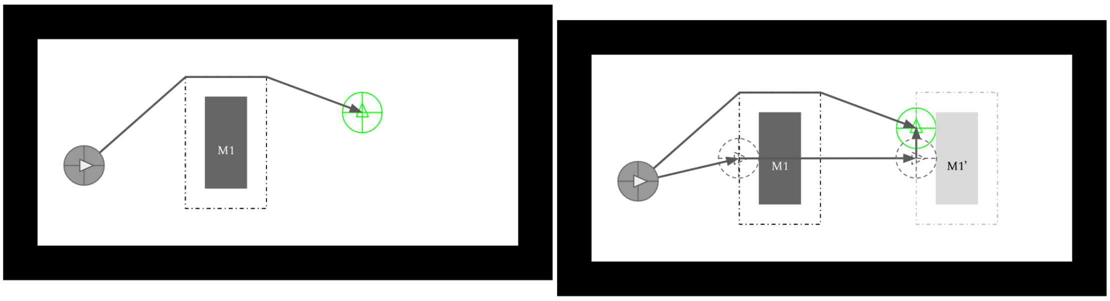
Third Optimization Only compute the third path component (from obstacle to goal) with A* if the simulated movement actually creates an opening (Figure 3.5b, line 7), since checking an opening creation is less computing time-consuming than running a search algorithm to the goal. However, this step also causes the loss of local optimality, because it prevents the algorithm to fully evaluate some plans that could improve the cost. Below are examples figures of cases where this happens, assuming the cost of following a path is the same whether it implies moving an obstacle or not. In Figure 3.3 and 3.4, no new opening is ever found, either because the blocking areas around the obstacle don't change (3.3) or because there were no blocking areas to begin with (3.4). Theses case are tentatively adressed in Chapter 4.



(A) Initial situation.

(B) Expected plan.

FIGURE 3.3: "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 original algorithm.

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

FIGURE 3.4: "Open space" case where the original algorithm will find only a suboptimal plan.

Fourth Optimization Finally, when the plan needs to be re-evaluated, new obstacles are evaluated first(Figure 3.5a, lines 8 to 12) and the resulting paths are saved into a list (Figure 3.5a, 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 3.5b, line 12). Then, this list is used to iterate over all obstacle/push direction combinations, and re-evaluate the corresponding plan (Figure 3.5a, 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 3.5b, 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."* [13]

Note on the use of A* in the main loop In Figure 3.5a, 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.

Algorithm 3 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

Algorithm 4 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 3.5: Optimized Algorithm as published by Wu et. al. in [31]

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 and the pseudocode formulation A.1 in Appendix A aim at fixing this.

A foreword on notations:

- Paths are ordered sets of "steps", which are themselves robot poses.
- Calling the A* or D*Lite algorithms returns A PATH. If no path is found, the returned set is empty: $\emptyset$.
- Plans are to be 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.

- Assuming we suppose a cost in distance, the norm of a path *path* (written $|path|$) corresponds to the sum of the euclidean distances between consecutive steps, and $+\infty$ if the path is empty.
- *moveCost* and *pushCost* are constants without dimension.
- The current robot pose is to be noted with an uppercase R, the initial pose with R_{init} and the goal pose with R_{goal}
- Obstacles are to be noted with a lowercase o. Lists or sets of obstacles will be noted with an uppercase O.

Note on update In the original paper, it is not explicit how the GET-NEW-INFORMATION method works. We therefore changed it to UPDATE-FROM-NEW-INFORMATION() method, 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 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.

Note on intersection detection The $p_{opt} \bigcap \mathcal{O}_{new} \neq \emptyset$ notation is used here as shorthand for checking that any of the new obstacles do not intersect with the robot's path. In implementation, this could be done for example by checking whether every pose in the path is not comprised in the inflated obstacles representation.

Note on re-evaluation The way the algorithm is written now, even new obstacles that have just been evaluated might be reevaluated, since they have been inserted into the P_{sort} list right before. Consequently, we should add a condition $p_{next}.o \notin \mathcal{O}_{new}$ around the line 28 of Algorithm 1 to further optimize the algorithm.

Note on getNext This is a helper method to traverse a list. It returns null when all elements have been traversed.

Note on stop condition The algorithm's stop condition is not explicit in none of the articles. If no path has been found even after considering all relevant obstacles, then the algorithm must return a success value.

Note on plan following In the article, the hypothesis is given that if the robot tries to move an obstacle but does not succeed, this obstacle will never be considered for manipulation again. It 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. This hypothesis is translated in pseudocode by replacing the vague "$R \leftarrow$ Next step in p_{opt}" statement 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.

Note on obstacle push pose For a given obstacle, the algorithm iterates over every push direction applicable to it, but doesn't iterate over every point 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 $o.init$ corresponds to the pose the robot must get into in order to move obstacle o in direction d. From the video[1] 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.

Note on $c_1 \neq \emptyset$ This condition is not in the original pseudocode, but is necessary to avoid the limit case where no path from the current robot pose to the obstacle is found, in which case, the manipulation plan cannot exist.

Note on bounding It is not said in the article how the "push on o in d possible" condition is verified, but we can assume that it is checked by verifying at each step that the obstacle's new occupied space doesn't intersect with any other obstacle. The $c_2 * pushCost$ bound here is not tight at all, which is fixed in the new optimization steps proposed in the second article. Still, it allows to cut down on unnecessary evaluations of extra pushes.

Note on $I.withSimulatedObstacleMove$ This notation is 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.

Note on elementary push It is not explicit in the article how an elementary $one\_push\_in\_d$ is computed, but we can assume it is the multiplication of a distance constant by the unit direction vector for the given direction d. We will make this more explicit in our own algorithm.

Note on opening detection Opening detection is not adressed in this paper and the method used is not explicit. A technical paper later written by co-authors Levihn and Stilman [17] however clears this ambiguity: *"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."*.

Note on c_2 As we only admit pushes in straight lines, and because the previous "push on o in d possible" condition means that there will be no collision on the manipulation path, c_2 simply is a path made from the pose to start moving the object and the pose where the robot leaves it.

[1] https://youtu.be/oQZLbJHYrl8

Note on $c_3 \neq \emptyset$ This condition is not in the original pseudocode, but is necessary to avoid the limit case where no path from the obstacle to the goal is found, in which case, the manipulation plan cannot exist.

3.3 Pseudocode expression of Levihn's recommendations

In the second article [13], 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 not affected by this optimization step. Our pseudocode formulation A.2 is available in Appendix A.

Second Optimization In this article, the authors precise that they are using an improved opening detection algorithm, detailed in their separate technical paper [17]. 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.

However, in the same way we proved that using opening detection for considering the computation of a full plan affected optimality before, since no measures are proposed in this new article to take this into account, we must assume that local optimality is not restored. In Chapter 4, we propose a measure for restoring optimality.

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 $|c_2|$ the cost of the first plan component c_1 and an underestimate of the cost of the third plan component c_3. 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. If not, $euCostL$ is used, the obstacle is re-evaluated and naturally added to the list ordered by $minCost$. This is achieved through the use of separate indexes for traversing the lists: i_e and i_m. This heuristic is invalidated anytime an obstacle has changed of place, potentially lowering c_2 or c_3. Thus, local optimality is no longer affected. For a more detailed explanation, please consult the following pseudocode or the original article [13].

Reordering the algorithm Since this is an interpretation, and for easier understanding, we took the liberty of cutting the "OPTIMIZED" algorithm (main loop) into two algorithms : the main loop where the plan is executed and knowledge about the environment updated (i.e. Algorithm 3: "MAKE-AND-EXECUTE-PLAN"), and the subroutine where the iteration over the obstacles is done to generate a new plan when necessary (i.e Algorithm 4: "MAKE-PLAN"). We also renamed the "OPT-EVALUATE-ACTION" subroutine (i.e Algorithm 2) into "PLAN-FOR-OBSTACLE" (i.e Algorithm 5). Note that the parameters p_{opt}, $euCostL$ and $minCostL$ are directly modified during the execution of the MAKE-PLAN() method, hence no return statement in it.

Note on D*Lite In the second article, it is mentionned twice that the path finding subroutine has been changed from A* to D*Lite. However, not even a hint of an explanation is given as to why this change, or what difference in the implementation it makes. Therefore, in our later final implementation, we shall stick with A*.

Note on update In addition to the previous note (3.2) We assume that I.freeSpaceCreated is True if any obstacle's occupied space has been reduced, False otherwise, and that I.allObstacles is the list of all observed obstacles in the current state.

Note on getting the list element and limit cases 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 "$minCostL[i_m]$ = null ? $+\infty$: $minCostL[i_m].minCost$") or implementing a custom array object with the wanted behaviour.

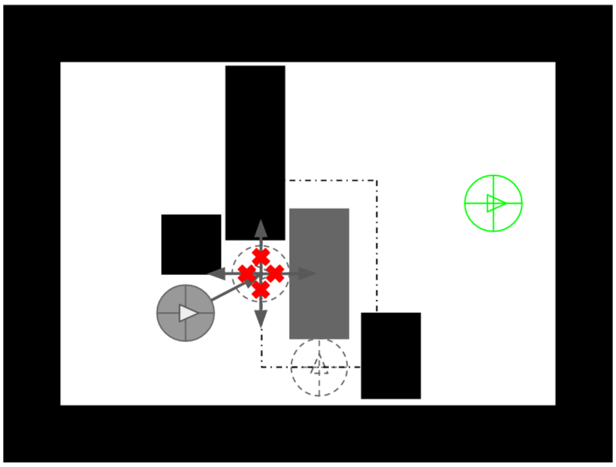
Note on *evaluatedObstacles* *evaluatedObstacles* is a set that remembers which obstacles have been evaluated in the current MAKE-PLAN instance. This is not mentioned in the original article, but it avoids evaluating an obstacle twice when it is added to $minCostL$.

Traversal note Stop condition: 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, it is not worth trying to evaluate any more options, and therefore the loop must end.

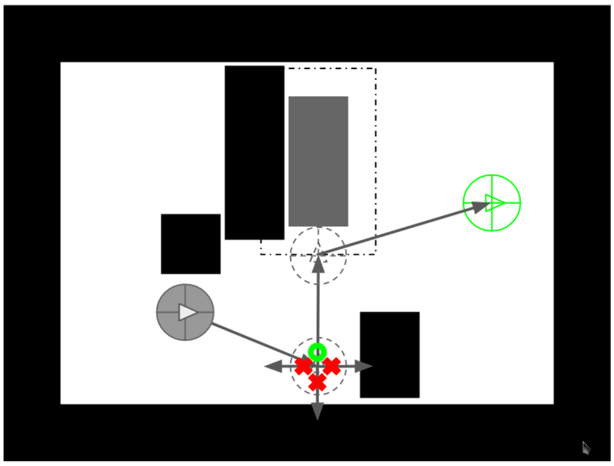
Note on priority If the current lowest cost entry is minCostL, evaluate the associated obstacle.

Note on postponing If i_e points to a lower cost than the one pointed by i_m, we only evaluate the associated object if $minCostL$ doesn't already contain an evaluation for it (because there is tighter bound for the cost of moving the object): the evaluation is thus postponed until the obstacle is reached through $minCostL$.

Note on manipulation points & c_1's computation 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 Note on obstacle push pose, 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 3.6: Illustration of the importance of considering all possibles manipulations for all manipulation poses and not just considering the nearest one.

Note on BA The BA variable in the OPT-EVALUATE-ACTION() function allows to remember the initial blocking areas when using the new algorithm for more efficient opening detection. On the first called, 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.

Note on allowed manipulations In the second article, the authors assert that they don't limit manipulation to pushes. However, it is clear, especially from the video, that manipulations are restricted to translations in a single direction.

Note on *seq* Seq corresponds to the number of unit translations that have been simulated.

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, ...), 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 carboard boxes as movable obstacles have shown that a good first approximation that guarantees quasi-systematic push manipulation successes are poses situated at the middle of the object's sides. 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).

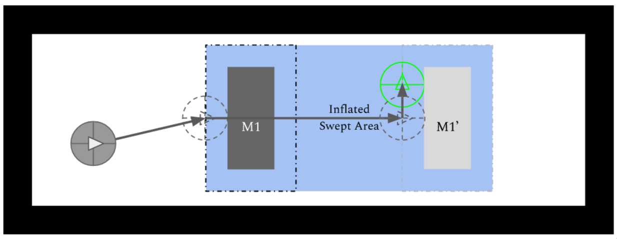
- **Obstacle discovery** As the robot approaches obstacles, their geometrical representation is updated according to what the robot's sensors can see. When executing a plan that includes the manipulation of an obstacle, said obstacle can actually change during the execution of the c_1 component, which 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.

Below, we propose, a way for restoring the optimality, assuming we are under the hypothesis of sole translations, in a single direction:

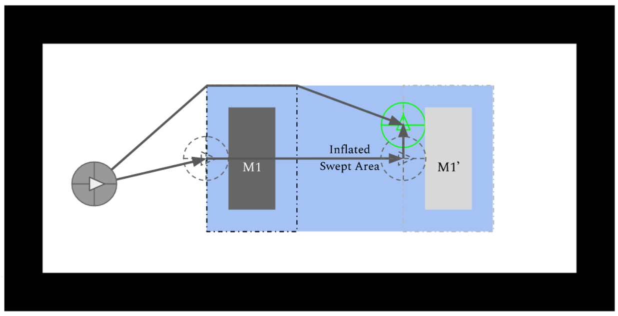
A new opening detection is defined by the disparition of at least one blocking area thanks to the considered manipulation [17]. A new opening is never detected if :

- Not a single blocking area disappears thanks to the considered manipulation, because the blocking areas do not vary enough or at all ("corridor" case),

- There are no blocking areas to begin with ("open space" case).



(A) "Corridor" case



(B) "Open space" case

FIGURE 4.1: Limit cases of the original algorithm with illustration of the "Inflated swept area"

In both cases, it is only interesting to consider the manipulation if it actually creates any chance of finding a path that has a lower cost than the one that avoids the obstacle. If no new opening is detected, and if, like in the "corridor" case, no path avoiding the obstacle was found, or, like in the "open space" case, a path 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 $goalPose \notin o.inflatedArea$

We have an intuition that these two extra verifications steps are sufficient to restore optimality, but this is no proper demonstration. 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, and postpone a proof to later work.

Note on 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).

Note on 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.

Note on [] and $\neq$ Here, the [] operator is 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).

Note on the use of $\bigcap$ 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

Note on the pseudocode The pseudocode discussed in this section and all following sections is based off the one in the first section of this chapter. For each proposition, we only modify the appropriate lines to properly highlight the indivual differences. In the last section, we merge all the propositions to give a global view of the proposed changes. The pseudocode formulation A.4 for this proposition is available in Appendix A.

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.

This identification of the obstacle's nature is supposed to be 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. However, robots come with all sort of sensors to detect obstacles: laser range finders, RGB(D) cameras, sonars, ... And often, as with Pepper, 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. 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,
- 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,
- 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-O1-C1 method, 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. 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}.

In order to keep our local optimality property, we also modified the main execution loop, see Algorithm 8. 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-O1-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, and allows to order them in a list $obsPoseL$, 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 $c_0 + c_1$ allowing not to evaluate many observation points.

Future work The computation and comparison if paths with observation poses may be further optimized by using a fitter algorithmic approach than calling A* as many times as needed, maybe using Dijkstra to first get the path to all observation points in a single graph search. The nature of the obstacle could also be used to adapt the way the obstacle is manipulated, for example by associating it to a specific maximum manipulation speed, which could depend on its physical characteristic.

4.3 Social awareness through placement consideration

The pseudocode formulation A.5 for this proposition is available in Appendix A.

Now that we have achieved basic capability for the algorithm to deal with obstacles depending on whether they have been explicitly defined as "movable" or not, we have a basic 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 must we 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.

In our proposition, we assume that "socially forbidden" areas are mapped in a similar fashion to the static obstacle map described before, with an arbitrary minimal integer value ALLOWED_VALUE and maximal value FORBIDDEN_VALUE. Since an obstacle may occupy several points, the total cost of moving an obstacle to some place is translated as the normalized sum of the costs of covering each point. If a point 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. On the contrary, if it is associated with the maximal value, no obstacle should ever be placed over it, raising the total cost to $+\infty$. This total cost is then simply added in the computation of the cost of the c_2 plan component as a product.

Future work Though we limit ourselves to a static map of "socially forbidden" areas in our implementation, 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 that depend on 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 the area they are about to traverse : 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

The pseudocode formulation A.6 for this proposition is available in Appendix A.

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 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. 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. 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 On experimentation repeatability with ROS and Pepper

5.2 Pushing tests with Pepper

5.3 ROS-Standards Compatible Simulator

5.4 Simulation results

Chapter 6

Conclusion and Perspectives

6.1 Contributions

6.2 Future work

Appendix A

Algorithms

A.1 Reworked Wu's algorithm

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

```
procedure OPTIMIZED(R_init, R_goal)
    R ← R_init
    blockedObsL ← ∅
    P_sort ← ∅
    O_new ← ∅
    isManipSuccess ← True
    I ← empty_occupation_grid
    p_opt.components ← [A*(R_init, R_goal, I)]
    p_opt.cost ← |p_opt.components[0]| * moveCost
    while R ≠ R_goal do
        UPDATE-FROM-NEW-INFORMATION(I)                         ▷ Note on update
        O_new ← O_new ⋃ I.newObstacles
        isPathFree ← p_opt ⋂ O_new ≠ ∅                          ▷ Note on intersection detection
        if not(isPathFree AND isManipSuccess) then
            p_opt.components ← [A*(R, R_goal, I)]
            p_opt.cost ← |p_opt.components[0]| * moveCost
            for each o ∈ O_new do
                for each possible push direction d on o do
                    p ← OPT-EVALUATE-ACTION(o, d, p_opt, I, R, R_goal, blockedObsL)
                    if p ≠ null then
                        P_sort.insert(p)
                    end if
                end for
            end for
            if P_sort ≠ ∅ then
                p_next ← P_sort[0]
                while p_opt.cost ≥ p_next.minCost AND p_next ≠ null do   ▷ Note on re-evaluation
                    p ← OPT-EVALUATE-ACTION(p_next.o, p_next.d, p_opt, I, R, R_goal, blockedObsL)
                    if p ≠ null AND p.cost ≤ p_opt.cost then
                        p_opt ← p
                    end if
                    p_next ← P_sort.getNext()                  ▷ Note on getNext
                end while
            end if
```

$\mathcal{O}_{new} \leftarrow \emptyset$
end if
if $p_{opt}.components = \emptyset$ **then**
return *False* ▷ Note on stop condition
end if
$isManipSuccess \leftarrow True$
$R_{next} \leftarrow p_{opt}$.getNextStep() ▷ Note on plan following
$c_{next} \leftarrow p_{opt}$.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
$P_{o,d} \leftarrow \emptyset$
$c_1 \leftarrow A^*(R, o.init, I)$ ▷ Note on obstacle push pose
if $c_1 = \emptyset$ **then** ▷ Note on $c_1 \neq \emptyset$
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** ▷ Note on bounding
$oSimPose \leftarrow oSimPose + one\_push\_in\_d$ ▷ Note on elementary push
if push created new opening **then** ▷ Note on opening detection
$c_2 \leftarrow \{o.init, oSimPose\}$ ▷ Note on c_2
$c_3 \leftarrow A^*(oSimPose, R_{goal}, I.withSimulatedObstacleMove)$ ▷ Note on *I.withSimulatedObstacleMove*
if $c_3 \neq \emptyset$ **then** ▷ Note on $c_3 \neq \emptyset$
$p.components \leftarrow [c_1, c_2, c_3]$
$p.cost \leftarrow (|c_1| + |c_3|) * moveCost + |c_2| * pushCost$
$p.minCost \leftarrow |c_2| * pushCost + |c_3| * moveCost$
$p.o, p.d \leftarrow o, d$
$P_{o,d} \leftarrow P_{o,d} \bigcup \{p\}$
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

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*Lite(R_{init}, R_{goal}, I)] ▷ Note on D*Lite
  $p_{opt}.cost \leftarrow |p_{opt}| * moveCost$
  while $R \neq R_{goal}$ **do**
    UPDATE-FROM-NEW-INFORMATION(I) ▷ Note on update
    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*Lite(R, R_{goal}, I)]
      $p_{opt}.cost \leftarrow |p_{opt}| * moveCost$
      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, 𝒪, blockedObsL, p_opt, euCostL, minCostL)
    for each o ∈ 𝒪 do
        C3_(Est) ← min({∀graspPoint ∈ o.graspPoints || |{graspPoint, R_goal}|})
        euCostL.insertOrUpdate({o, C3_(Est)})
    end for
    i_e, i_m ← 0, 0                                  ▷ Note on getting the list element and limit cases
    evaluatedObstacles ← ∅                            ▷ Note on evaluatedObstacles
    while min(minCostL[i_m].minCost, euCostL[i_e].c3_est) < p_opt.cost do      ▷ Traversal note
        if minCostL[i_m].minCost < euCostL[i_e].c3_est then                    ▷ Note on priority
            o ← minCostL[i_m].obstacle
            if o ∉ evaluatedObstacles then
                p ← PLAN-FOR-OBSTACLE(o, p_opt, I, R, R_goal, blockedObsL)
                if p ≠ null then
                    minCostL.insertOrUpdate({o, p.minCost})
                else
                    minCostL.insertOrUpdate({o, +∞})
                end if
                evaluatedObstacles.insert(o)
            end if
            i_m ← i_m + 1
        else
            if not minCostL.contains(euCostL[i_e].obstacle) then               ▷ Note on postponing
                o ← euCostL[i_e].obstacle
                if o ∉ evaluatedObstacles then
                    p ← PLAN-FOR-OBSTACLE(o, p_opt, I, R, R_goal, blockedObsL)
                    if p ≠ null then
                        minCostL.insertOrUpdate({o, p.minCost})
                    else                        ▷ Corresponds to the "If p ≠ null" statement.
                        minCostL.insertOrUpdate({o, +∞})
                    end if
                    evaluatedObstacles.insert(o)
                end if
            end if
            i_e ← 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 ∈ blockedObsL then
        return null
    end if
    P_{o,d} ← ∅
    c_1 ← D*Lite(R, o.init, I)                          ▷ Note on manipulation points & c_1's computation
    if c_1 = ∅ then
        return null
    end if
    BA ← null                                           ▷ Note on BA
    for each possible manipulation direction d on o do  ▷ Note on allowed manipulations
        seq ← 1                                         ▷ Note on seq
        oSimPose ← o.init + one_translation_in_d
        c3_(Est) ← {oSimPose, R_goal}
        C_est ← (|c_1| + |c3_(Est)|) * moveCost + seq * one_translation_in_d * pushCost
        while C_est ≤ 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 ← {o.init, oSimPose}
                c_3 ← D*Lite(oSimPose, R_goal, I.withSimulatedObstacleMove)
                if c_3 ≠ ∅ then
                    p.components ← [c_1, c_2, c_3]
                    p.cost ← (|c_1| + |c_3|) * moveCost + |c_2| * pushCost
                    p.minCost ← |c_2| * pushCost + |c_3| * moveCost
                    p.o, p.d ← o, d
                    P_{o,d} ← P_{o,d} ∪ {p}
                    if p.cost < p_opt.cost then
                        p_opt ← p
                    end if
                end if
            end if
            seq ← seq + 1
            oSimPose ← oSimPose + one_translation_in_d
            c3_(Est) ← {oSimPose, R_goal}
            C_est ← (|c_1| + |c3_(Est)|) * moveCost + seq * one_translation_in_d * pushCost
        end while
    end for
    return p ∈ P_{o,d} with minimal p.cost or null if P_{o,d} = ∅
end procedure
```

A.3 Algorithm adapted to our use case

Algorithm 6 Execution loop taking our hypotheses into account.

procedure MAKE-AND-EXECUTE-PLAN(R_{init}, R_{goal}, I_{init})
    ... ▷ Initialization (lines 2 to 6 in Algorithm 3)
    $I \leftarrow I_{init}$
    $p_{opt}.components \leftarrow [A^*(R_{init}, R_{goal}, I)]$
    $p_{opt}.cost \leftarrow |p_{opt}| * moveCost$
    while $R \neq R_{goal}$ **do**
        UPDATE-FROM-NEW-INFORMATION(I)
        if I.freeSpaceCreated **then**
            $minCostL \leftarrow \emptyset$
        end if
        $\mathcal{O}_{new} \leftarrow \mathcal{O}_{new} \bigcup I.newObstacles$
        $\mathcal{O} \leftarrow I.allObstacles$
        $isPathFree \leftarrow p_{opt} \bigcap \mathcal{O}_{new} \neq \emptyset$
        $isPushPoseValid \leftarrow True$
        $isManipSafe \leftarrow True$
        $isObstacleSame \leftarrow True$
        if $p_{opt}.o$ exists **then** ▷ If p_{opt} includes the manipulation of an obstacle.
            $isObstacleSame \leftarrow \mathcal{O}[p_{opt}.o] = p_{opt}.o$ ▷ Note on [] and $\neq$
            if not $isObstacleSame$ **then**
                $p_{opt}.o \leftarrow \mathcal{O}[p_{opt}.o.id]$ ▷ Update the copy.
                $p_{opt}.safeSweptArea \leftarrow$ 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** ▷ Note on the use of $\bigcap$
                $isManipSafe \leftarrow False$
            end if
        end if
        if not($isPathFree$ AND $isManipSuccess$ AND $isManipSafe$ AND $isPushPoseValid$) **then**
            $p_{opt}.components \leftarrow [A^*(R, R_{goal}, I)]$
            $p_{opt}.cost \leftarrow |p_{opt}| * moveCost$
            MAKE-PLAN(R, R_{goal}, I, $\mathcal{O}$, $blockedObsL$, p_{opt}, $euCostL$, $minCostL$)
            $\mathcal{O}_{new} \leftarrow \emptyset$
        end if
        ... ▷ Execution (lines 24 to 35 in Algorithm 3)
    end while
    return $True$
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 ∈ blockedObsL then
        return null
    end if
    P_o,d ← ∅
    for each pushPose in o.pushPoses do
        pushUnit ← (cos(pushPose.yaw), sin(pushPose.yaw))   ▷ Unit vector for push direction
        c1 ← A*(R, pushPose, I)                              ▷ c1 is computed for each push pose
        if c1 = ∅ then
            continue                                          ▷ Note on continue
        end if
        seq ← 1
        translation ← pushUnit * onePushDist * seq           ▷ onePushDist is a distance constant
        safeSweptArea ←GET-SAFE-SWEPT-AREA(o, translation, I)
        oSimPose ←pushPose + translation
        c3_(Est) ← {oSimPose, R_goal}
        C_est ← (|c1| + |c3_(Est)|) * moveCost+|translation| * o.pushCost
        while C_est ≤ p_opt.cost AND safeSweptArea ≠ null do
            c2 ← {pushPose, oSimPose}
            c3 ← A*(oSimPose, R_goal, I.withSimulatedObstacleMove)
            if c3 ≠ ∅ then
                p.components ← [c1, c2, c3]
                p.cost ← (|c1| + |c3|) * moveCost + |c2|*o.pushCost
                p.minCost ← |c2|*o.pushCost+|c3| * moveCost
                p.o ← COPY(o)                                 ▷ Note on COPY
                p.translation ← translation                   ▷ Note on saving translation in p_opt
                p.safeSweptArea ← safeSweptArea
                P_o,d ← P_o,d ∪ {p}
                if p.cost < p_opt.cost then
                    p_opt ← p
                end if
            end if
            seq ← seq + 1
            translation ← pushUnit * onePushDist * seq
            safeSweptArea ←GET-SAFE-SWEPT-AREA(o, translation, I)
            oSimPose ←pushPose + translation
            c3_(Est) ← {oSimPose, R_goal}
            C_est ← (|c1| + |c3_(Est)|) * moveCost+|translation| * o.pushCost
        end while
    end for
    return p ∈ P_o,d with minimal p.cost or null if P_o,d = ∅
end procedure
```

A.4 Algorithm proposition: Social awareness through manipulation authorization consideration

Algorithm 8 Execution loop modified for allowing observation.

procedure MAKE-AND-EXECUTE-PLAN(R_{init}, R_{goal}, I_{init})
    ... ▷ Initialization (lines 2 to 5 in Algorithm 6)
    $isObservable \leftarrow True$
    while $R \neq R_{goal}$ **do**
        UPDATE-FROM-NEW-INFORMATION(I)
        ... ▷ Knowledge update and checks (lines 7 to 29 in Algorithm 6)
        if not($isPathFree$ AND $isManipSuccess$ AND $isManipSafe$ AND $isPushPoseValid$ AND $isObservable$) **then**
            $p_{opt}.components \leftarrow [A^*(R, R_{goal}, I)]$
            $p_{opt}.cost \leftarrow |p_{opt}| * moveCost$
            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
        $isManipSuccess \leftarrow True$
        $isObservable \leftarrow True$
        $R_{next} \leftarrow p_{opt}$.getNextStep()
        $c_{next} \leftarrow p_{opt}$.getNextStepComponent()
        if $p_{opt}.o \neq$ null $p_{opt}.o.movableStatus = IS\_MAYBE\_MOVABLE$ AND **not** $isObstacleSame$ AND ($c_{next} = c_0$ OR $c_{next} = c_1$) **then**
            $fObsPose \leftarrow$ GET-FIRST-PATH-OBSPOSE($p_{opt}.o$, p_{opt}.get-c_0() + p_{opt}.get-c_1(), I)
            if $fObsPose =$ null **then**
                $isObservable \leftarrow False$
                continue
            end if
        end if
        $R_{real} \leftarrow$ ROBOT-GOTO(R_{next})
        if $c_{next} = c_2$ 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 9 Obstacle evaluation subroutine modified for allowing observation.

```
procedure PLAN-FOR-OBSTACLE(o, p_opt, I, R, R_goal, blockedObsL)
    if o ∈ blockedObsL OR o.movableStatus = IS_NOT_MOVABLE then
        return null
    end if
    P_{o,d} ← ∅
    for each pushPose in o.pushPoses do
        pushUnit ← (cos(pushPose.yaw), sin(pushPose.yaw))
        c_1 ← A*(R, pushPose, I)
        if c_1 = ∅ then
            continue
        end if
        c_0 ← ∅
        if o.movableStatus = IS_MAYBE_MOVABLE then
            obsPose ← GET-FIRST-PATH-OBSPOSE(o, c_1, I)
            if obsPose ≠ null then
                c_0, c_1 ← c_1[c_1.firstPose : obsPose], c_1[obsPose : c_1.lastPose]
            else
                COMPUTE-O1-C1(o, I, R, pushPose, c_0, c_1)
                if c_0 = ∅ OR c_1 = ∅ then
                    continue
                end if
            end if
        end if
        seq ← 1
        translation ← pushUnit * onePushDist * seq
        safeSweptArea ←GET-SAFE-SWEPT-AREA(o, translation, I)
        oSimPose ← pushPose + translation
        c_{3(Est)} ← {oSimPose, R_goal}
        C_est ← ((c_0 ≠ ∅?|c_0| : 0)+|c_1| + |c_{3(Est)}|) * moveCost + |translation| * o.pushCost
        while C_est ≤ p_opt.cost AND safeSweptArea ≠ null do
            c_2 ← {pushPose, oSimPose}
            c_3 ← A*(oSimPose, R_goal, I.withSimulatedObstacleMove)
            if c_3 ≠ ∅ then
                p.components ← c_0 ≠ ∅ ? [c_0, c_1, c_2, c_3] : [c_1, c_2, c_3]
                p.cost ← ((c_0 ≠ ∅?|c_0| : 0)+|c_1| + |c_3|) * moveCost + |c_2| * o.pushCost
                p.minCost ← |c_2| * o.pushCost + |c_3| * moveCost
                ...            ▷ Affectation of other variables of p (lines 25 to 31 in Algorithm 7)
            end if
            seq ← seq + 1
            translation ← pushUnit * onePushDist * seq
            safeSweptArea ←GET-SAFE-SWEPT-AREA(o, translation, I)
            oSimPose ← pushPose + translation
            c_{3(Est)} ← {oSimPose, R_goal}
            C_est ← ((c_0 ≠ ∅?|c_0| : 0)+|c_1| + |c_{3(Est)}|) * moveCost + |translation| * o.pushCost
        end while
    end for
    return p ∈ P_{o,d} with minimal p.cost or null if P_{o,d} = ∅
end procedure
```

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

```
procedure GET-FIRST-PATH-OBSPOSE(o, path, I)
    for each pose in path do
        if IS-OBS-IN-FOV-FOR-POSE(o, pose, I) then
            return pose
        end if
    end for
    return null
end procedure
```

Algorithm 11 Subroutine for computing c_0 and c_1 if c_1 is not already valid.

```
procedure COMPUTE-O1-C1(o, I, R, pushPose, c_0, c_1)
    c_1 ← ∅
    totalCost ← +∞
    for each obsPose in o.obsPoses do
        o ← A*(R, obsPose, I)
        c ← A*(obsPose, pushPose, I)
        newTotalCost = |o| + |c|
        if newTotalCost < +∞ AND (newTotalCost < totalCost OR (newTotalCost = totalCost AND |o| < |c_0|) then
            c_0 = o
            c_1 = c
            totalCost = |c_0| + |c_1|
        end if
    end for
end procedure
```

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

procedure OPT-COMPUTE-O1-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**
      $o \leftarrow A^*(R, op_{next}.obsPose, I)$
      $c \leftarrow A^*(op_{next}.obsPose, pushPose, I)$
      $newTotalCost = |o| + |c|$
      if $newTotalCost < +\infty$ AND ($newTotalCost < totalCost$ OR ($newTotalCost = totalCost$ AND $|o| < |c_0|$) **then**
        $c_0 = o$
        $c_1 = c$
        $totalCost = |c_0| + |c_1|$
      end if
      $op_{next} \leftarrow euPosesCostL$.getNext()
    end while
  end if
end procedure

A.5 Algorithm proposition: Social awareness through placement consideration

Algorithm 13 Obstacle evaluation subroutine modified for considering placement.

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

Algorithm 14 Obstacle evaluation subroutine modified for considering placement.

procedure GET-OCC-COST($simOccPoints$, $occCostGrid$)
    $VALUE\_RANGE \leftarrow (FORBIDDEN\_VALUE - ALLOWED\_VALUE)$
    $cost \leftarrow 1$
    for each $point$ in $simOccPoints$ **do**
        $valueForPoint \leftarrow occCostGrid[point]$
        if $valueForPoint = FORBIDDEN\_VALUE$ **then**
            return $+\infty$
        else if $valueForPoint = ALLOWED\_VALUE$ **then**
            $valueForPoint \leftarrow 0$
        end if
        $cost \leftarrow cost + valueForPoint / VALUE\_RANGE$
    end for
    return $cost$
end procedure

A.6 Algorithm proposition: Taking dynamic obstacles into account

Algorithm 15 Execution loop taking dynamic obstacles into account.

procedure MAKE-AND-EXECUTE-PLAN(R_{init}, R_{goal}, I_{init})
    ... ▷ Initialization (lines 2 to 5 in Algorithm 6)
    $isBlockingObsMoved \leftarrow False$
    while $R \neq R_{goal}$ **do**
        UPDATE-FROM-NEW-INFORMATION(I)
        if I.freeSpaceCreated **then**
            $minCostL \leftarrow \emptyset$
        end if
        $\mathcal{O} \leftarrow I.allObstacles$
        $isPathFree \leftarrow p_{opt} \bigcap \mathcal{O} \neq \emptyset$
        $isBlockingObsMoved \leftarrow I.movedObstacles \neq \emptyset$
        ... ▷ Plan validity checks (lines 14 to 29 in Algorithm 6)
        $isPlanValid \leftarrow$ ($isPathFree$ AND $isManipSuccess$ AND $isManipSafe$ AND $isPushPoseValid$)
        if not $isPlanValid$ **then**
            $p_{opt}.components \leftarrow [A^*(R, R_{goal}, I)]$
            $p_{opt}.cost \leftarrow |p_{opt}| * moveCost$
        end if
        if not $isPlanValid$ OR ($isBlockingObsMoved$) **then**
            MAKE-PLAN(R, R_{goal}, I, $\mathcal{O}$, $blockedObsL$, p_{opt}, $euCostL$, $minCostL$)
            $isManipSuccess \leftarrow True$ ▷ Line 27 in Algorithm 3 is moved here.
            continue
        end if
        ... ▷ Execution (lines 24 to 26 and 28 to 35 in Algorithm 3)
        ... ▷ Execution (lines 24 to 35 in Algorithm 3)
    end while
    return $True$
end procedure

A.7 Merged proposition algorithm

Algorithm 16 Merged execution loop.

procedure MAKE-AND-EXECUTE-PLAN(R_{init}, R_{goal}, I_{init})
  $R \leftarrow R_{init}$
  $\mathcal{O}_{new} \leftarrow \emptyset$
  $isManipSuccess \leftarrow True$
  $blockedObsL \leftarrow \emptyset$
  $euCostL, minCostL \leftarrow \emptyset, \emptyset$
  $I \leftarrow I_{init}$
  $p_{opt}.components \leftarrow [A^*(R_{init}, R_{goal}, I)]$
  $p_{opt}.cost \leftarrow |p_{opt}| * moveCost$
  $isObservable \leftarrow True$
  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**
      $p_{opt}.components \leftarrow [A^*(R, R_{goal}, I)]$
      $p_{opt}.cost \leftarrow |p_{opt}| * moveCost$
    end if
    if not $isPlanValid$ OR ($isBlockingObsMoved$) **then**
      MAKE-PLAN(R, R_{goal}, I, $\mathcal{O}$, $blockedObsL$, p_{opt}, $euCostL$, $minCostL$)
      $isManipSuccess \leftarrow True$
      continue
    end if

if $p_{opt}.components = \emptyset$ **then**
  return *False*
end if
$isManipSuccess \leftarrow True$
$isObservable \leftarrow True$
$R_{next} \leftarrow p_{opt}$.getNextStep()
$c_{next} \leftarrow p_{opt}$.getNextStepComponent()
if $p_{opt}.o \neq$ null $p_{opt}.o.movableStatus = IS\_MAYBE\_MOVABLE$ AND **not** *isObstacleSame* AND ($c_{next} = c_0$ OR $c_{next} = c_1$) **then**
  $fObsPose \leftarrow$ GET-FIRST-PATH-OBSPOSE($p_{opt}.o$, p_{opt}.get-c_0() + p_{opt}.get-c_1(), I)
  if $fObsPose =$ null **then**
    $isObservable \leftarrow False$
    continue
  end if
end if
$R_{real} \leftarrow$ ROBOT-GOTO(R_{next})
if $c_{next} = c_2$ 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 17 Merged obstacle evaluation subroutine

```
procedure PLAN-FOR-OBSTACLE(o, p_opt, I, R, R_goal, blockedObsL)
    if o ∈ blockedObsL OR o.movableStatus = IS_NOT_MOVABLE then
        return null
    end if
    P_{o,d} ← ∅
    for each pushPose in o.pushPoses do
        pushUnit ← (cos(pushPose.yaw), sin(pushPose.yaw))
        c_1 ← A*(R, pushPose, I)
        if c_1 = ∅ then
            continue
        end if
        c_0 ← ∅
        if o.movableStatus = IS_MAYBE_MOVABLE then
            obsPose ← GET-FIRST-PATH-OBSPOSE(o, c_1, I)
            if obsPose ≠ null then
                c_0, c_1 ← c_1[c_1.firstPose : obsPose], c_1[obsPose : c_1.lastPose]
            else
                OPT-COMPUTE-O1-C1(o, I, R, pushPose, c_0, c_1)
                if c_0 = ∅ OR c_1 = ∅ then
                    continue
                end if
            end if
        end if
        seq ← 1
        translation ← pushUnit ∗ onePushDist ∗ seq
        safeSweptArea ←GET-SAFE-SWEPT-AREA(o, translation, I)
        oSimPose ← pushPose + translation
        suppC_M ←GET-OCC-COST(GET-OBS-POINTS(o, translation), occCostGrid)
        c3_(Est) ← {oSimPose, R_goal}
        C_est ← ((c_0 ≠ ∅?|c_0| : 0) + |c_1| + |c3_(Est)|) ∗ moveCost + |translation| ∗ o.pushCost ∗ suppC_M
```

while $C_{est} \leq p_{opt}.cost$ AND $safeSweptArea \neq$ null **do**
$c_2 \leftarrow \{pushPose, oSimPose\}$
$c_3 \leftarrow A^*(oSimPose, R_{goal}, I.withSimulatedObstacleMove)$
if $c_3 \neq \emptyset$ **then**
$p.components \leftarrow c_0 \neq \emptyset\ ?\ [c_0, c_1, c_2, c_3] : [c_1, c_2, c_3]$
$p.cost \leftarrow ((c_0 \neq \emptyset ? |c_0| : 0) + |c_1| + |c_3|) * moveCost + |c_2| * o.pushCost * suppC_M$
$p.minCost \leftarrow |c_2| * o.pushCost * suppC_M + |c_3| * moveCost$
$p.o \leftarrow$ COPY(o)
$p.translation \leftarrow translation$
$p.safeSweptArea \leftarrow safeSweptArea$
$P_{o,d} \leftarrow P_{o,d} \bigcup \{p\}$
if $p.cost < p_{opt}.cost$ **then**
$p_{opt} \leftarrow p$
end if
end if
$seq \leftarrow seq + 1$
$translation \leftarrow pushUnit * onePushDist * seq$
$safeSweptArea \leftarrow$GET-SAFE-SWEPT-AREA(o, $translation$, I)
$oSimPose \leftarrow pushPose + translation$
$suppC_M \leftarrow$GET-OCC-COST(GET-OBS-POINTS(o, $translation$), $occCostGrid$)
$c_{3_{(Est)}} \leftarrow \{oSimPose, R_{goal}\}$
$C_{est} \leftarrow ((c_0 \neq \emptyset ? |c_0| : 0) + |c_1| + |c_{3_{(Est)}}|) * moveCost + |translation| * o.pushCost * suppC_M$
end while
end for
return $p \in P_{o,d}$ with minimal $p.cost$ or null if $P_{o,d} = \emptyset$
end procedure

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

procedure ASSIGN-NR(BA, x, y, $index$)
  for i from -1 to 1 **do** ▷ Assignment is performed based on the 3*3 neighborhood.
    for j from -1 to 1 **do**
      if $BA[x+i][y+j] \neq 0$ **then** ▷ If a number is already in the neighborhood, ...
        $BA[x][y] \leftarrow BA[x+i][y+j]$ ▷ ... the same number is assigned to the element.
        return
      end if
    end for
  end for
  $BA[x][y] \leftarrow index$ ▷ Else a new number is assigned, equal to the new index.
  $index \leftarrow index + 1$ ▷ Only increment index if new intersection is created.
  return
end procedure

procedure COMPARE(BA, BA_s^*) ▷ This function checks for non-zero entries in both matrices.
  $BS \leftarrow$ COPY(BA)
  $del_{num} \leftarrow \emptyset$ ▷ Set of the indexes of obstacles to delete from BS.
  for x from 0 to $|BS|$ **do** ▷ Iterate over BS.
    for y from 0 to $|BS[x]|$ **do**
      if $BA[x][y] \in del_{num}$ **then** ▷ Note on deletion
        $BS[x][y] \leftarrow 0$
      end if
      if $BA[x][y] \neq 0$ AND $BA_s^*[x][y] \neq 0$ **then** ▷ Note on deletion
        $del_{num} = del_{num} \bigcup BS[x][y]$
        $BS[x][y] \leftarrow 0$
      end if
    end for
  end for
  return BS
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							
			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]
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]
	2D costmap	[j], [k]		[j]	[k]			[j]		[j] [k]
	3D metric map	[a], [c], [f], [g]	[c]				[f] [g]	[c]		[c] [g]
	Complete	[a], [b], [d], [h], [l]	[b] [d]			[d]	[h] [l]	[b]	[d]	[b] [d] [l]
	Partial	[c], [g], [Exp]	[c] [Exp]				[g]	[c] [Exp]		[c] [g] [Exp]
	Unknown	[e], [i], [f], [j], [k]	[e]	[j]	[i] [k]		[f]	[e] [i] [j]		[e] [i] [j] [k]
	Perfect data	[a], [b], [d], [e], [i], [Exp]	[b] [d] [e] [Exp]		[i]	[d]		[b] [e] [i] [Exp]	[d]	[b] [d] [e] [i] [Exp]
	Approximative data	[c], [f], [g], [h], [j], [k], [l]	[c]	[j]	[k]		[f] [g] [h] [l]	[c] [j]		[c] [g] [j] [k] [l]
	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]
Obstacle characteristics	Naive 2D projection	[a], [b], [e], [g], [h], [i], [l]	[b] [e]		[i]		[g] [h] [l]	[b] [e] [i]		[b] [e] [g] [i] [l]
	2D Projection using Convex-Hull	[c], [d], [f], [Exp]	[c] [d] [Exp]			[d]	[f]	[c] [Exp]	[d]	[c] [d] [Exp]
	Any obstacle types	[f], [g], [h], [i], [j], [k], [l]		[j]	[i] [k]		[f] [g] [h] [l]	[i] [j]		[g] [i] [j] [k] [l]
	Only polygonal obstacles	[a], [b], [c], [d], [Exp]	[b] [c] [d] [Exp]			[d]		[b] [c] [Exp]	[d]	[b] [c] [d] [Exp]
	Only rectangular obstacles	[e]	[e]					[e]		[e]
	Human obstacle									
	Moving obstacle	[k], [Exp]	[Exp]		[k]			[Exp]		[k] [Exp]
	Metadata on obstacle's physics	[a], [b], [c], [d], [h]	[b] [c] [d]			[d]	[h]	[b] [c]	[d]	[b] [c] [d]
	Obstacle can be translated in 2D plane	[a], [b], [c], [d], [e], [f], [g], [h], [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]
	Translation limited to the 2D plane axes	[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]
Robot characteristics	HRP2 Robot	[a], [c], [f]	[c]				[f]	[c]		[c]
	PR2 Robot	[g]					[g]			[g]
	GOLEM Krang Robot	[h], [l]					[h] [l]			[l]
	Custom robot vehicle for MAGIC 2010 Competition	[j], [k]		[j]	[k]			[j]		[j] [k]
	Pepper Robot	[Exp]	[Exp]					[Exp]		[Exp]
	Nondescript humanoid robot	[b], [d]	[b] [d]			[d]		[b]	[d]	[b] [d]
	Nondescript wheeled robot	[e], [i]	[e]		[i]			[e] [i]		[e] [i]
	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]
	Unlimited field of vision	[a], [b], [c], [d], [h], [l]	[b] [c] [d]			[d]	[h] [l]	[b] [c]	[d]	[b] [c] [d] [l]
	Robot can translate on the plane	[a], [b], [c], [d], [e], [f], [g], [h], [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]
	Robot can rotate in the plane	[a], [b], [c], [d], [e], [f], [g], [h], [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]
	Lift & Drop	[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]
	Push	[b], [c], [d], [e], [f], [g], [h], [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]
Problem class	L1	[a], [b], [c], [e], [f], [h], [i], [j], [k], [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]
	LkM	[d], [g]	[d]			[d]	[g]		[d]	[d] [g]

			Evaluation and evolution of an obstacle's "movable" characteristic and its associated cost					Object manipulation maneuver planning			Planning taking uncertainty into account					
			"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
			[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], [l], [Exp]	[j], [k]	[c], [g], [h], [j], [k], [l]	[g], [j], [k]	[g]	[h], [l]	[l]	[f]
Knowledge of the environment	2D metric map	[b], [d], [e],[h], [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] [k]				[j] [k]	[j] [k]	[j] [k]				
	3D metric map	[a], [c], [f], [g]	[f] [g]	[a] [c]				[c]	[a] [c] [f] [g]		[c] [g]	[g]	[g]			[f]
	Complete	[a], [b], [d], [h], [l]	[h] [l]	[a] [b]				[h] [l]	[a] [b] [d] [h] [l]		[h] [l]			[h] [l]	[l]	
	Partial	[c], [g], [Exp]	[g] [Exp]	[c]	[Exp]			[c]	[c] [g] [Exp]		[c] [g]	[g]	[g]			
	Unknown	[e], [i], [f], [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]	[e] [i] [Exp]	[a] [b]	[e] [i] [Exp]		[i]		[a] [b] [d] [e] [Exp]							
	Approximative data	[c], [f], [g], [h], [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] [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]	[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]	[f] [Exp]	[c]	[Exp]			[c]	[c] [d] [f] [Exp]		[c]					[f]
	Any obstacle types	[f], [g], [h], [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] [l]	[g] [j] [k]	[g]	[h] [l]	[l]	[f]
	Only polygonal obstacles	[a], [b], [c], [d], [Exp]	[Exp]	[a] [b] [c]	[Exp]			[c]	[a] [b] [c] [d] [Exp]		[c]					
	Only rectangular obstacles	[e]	[e]		[e]				[e]							
	Human obstacle															
	Moving obstacle	[k], [Exp]	[k] [Exp]		[Exp]	[k]			[Exp]	[k]	[k]	[k]				
	Metadata on obstacle's physics	[a], [b], [c], [d], [h]	[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], [h], [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] [l] [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]							
	Obstacle can be rotated in the normal to the 2D plane	[a], [b], [d], [g], [h], [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]	[f]	[a] [c]				[c]	[a] [c] [f]		[c]					[f]
	PR2 Robot	[g]	[g]						[g]		[g]	[g]	[g]			
	GOLEM Krang Robot	[h], [l]	[h] [l]					[h] [l]	[h] [l]		[h] [l]			[h] [l]	[l]	
	Custom robot vehicle for MAGIC 2010 Competition	[j], [k]	[j] [k]			[j] [k]				[j] [k]	[j] [k]	[j] [k]				
	Pepper Robot	[Exp]	[Exp]		[Exp]				[Exp]							
	Nondescript humanoid robot	[b], [d]		[b]					[b] [d]							
	Nondescript wheeled robot	[e], [i]	[e] [i]		[e] [i]		[i]		[e]							
	Limited field of vision	[e], [f], [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]	[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], [h], [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] [l] [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], [h], [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] [l] [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]	[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], [h], [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] [h] [l] [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], [h], [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] [l] [Exp]	[j] [k]	[c] [h] [j] [k] [l]	[j] [k]		[h] [l]	[l]	[f]
	LkM	[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], [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], [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] [k] [l] [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] [l] [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]***	***[f] [g] [j] [k] [l] [Exp]***	[a] [c]	***[Exp]***	[j] [k]		[c] [l]	***[a] [c] [d] [f] [g] [l] [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] 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.

[2] 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).

[3] 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.

[4] 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.

[5] 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.

[6] 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).

[7] 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).

[8] 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.

[9] 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.

[10] 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.

[11] 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).

[12] M. Levihn, J. Scholz, and M. Stilman. "Planning with movable obstacles in continuous environments with uncertain dynamics". In: *2013 IEEE International Conference on Robotics and Automation*. 2013 IEEE International Conference on Robotics and Automation. May 2013, pp. 3832–3838. DOI: 10.1109/ICRA.2013.6631116.

[13] 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.

[14] 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.

[15] 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.

[16] 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).

[17] 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).

[18] 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).

[19] 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.

[20] 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.

[21] 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.

[22] 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).

[23] 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.

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

[25] 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.

[26] 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.

[27] 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).

[28] 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).

[29] 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).

[30] 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).

[31] 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.