Navigation Among Movable Obstacles (NAMO) Extended to Social and Multi-Robot Constraints

Benoit Renault

N°d'ordre NNT : 2023ISAL0105

THESE de DOCTORAT DE L'INSA LYON, membre de l'Université de Lyon

Ecole Doctorale N° 512
Informatique et Mathématiques

Spécialité/ discipline de doctorat :

Informatique

Soutenue publiquement le 19/12/2023, par :
Benoit RENAULT

NAvigation en milieu MOdifiable (NAMO) étendue à des contraintes sociales et multi-robots

---

Devant le jury composé de :

Dugdale, Julie	Professeur des Universités	Université de Grenoble, LIG	Président.e
Mathieu, Philippe	Professeur des Universités	Université de Lille, CRISTAL	Rapporteur
Michel, Fabien	Maître de Conférences HDR	Université Montpellier 2, LIRMM	Rapporteur
Alami, Rachid	Directeur de Recherche émérite CNRS	Toulouse, LAAS	Examinateur
Simonin, Olivier	Professeur des Universités	INSA-Lyon, CITI	Directeur de thèse
Saraydaryan, Jacques	Enseignant Chercheur CPE Lyon	CITI	Co-encadrant

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Département de la Formation par la Recherche et des Études Doctorales (**FEDORA**)

Bâtiment INSA direction, 1er étage
37, av. J. Capelle
69621 Villeurbanne Cédex
fedora@insa-lyon.fr

Référence : TH1046_RENAULT Benoit

L'INSA Lyon a mis en place une procédure de contrôle systématique via un outil de détection de similitudes (logiciel Compilatio). Après le dépôt du manuscrit de thèse, celui-ci est analysé par l'outil. Pour tout taux de similarité supérieur à 10%, le manuscrit est vérifié par l'équipe de FEDORA. Il s'agit notamment d'exclure les auto-citations, à condition qu'elles soient correctement référencées avec citation expresse dans le manuscrit.

Par ce document, il est attesté que ce manuscrit, dans la forme communiquée par la personne doctorante à l'INSA Lyon, satisfait aux exigences de l'Etablissement concernant le taux maximal de similitude admissible.

Département FEDORA – INSA Lyon - Ecoles Doctorales

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	**CHIMIE DE LYON** **https://www.edchimie-lyon.fr** Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage secretariat@edchimie-lyon.fr	**M. Stéphane DANIELE** C2P2-CPE LYON-UMR 5265 Bâtiment F308, BP 2077 43 Boulevard du 11 novembre 1918 69616 Villeurbanne **directeur@edchimie-lyon.fr**
E.E.A.	**ÉLECTRONIQUE, ÉLECTROTECHNIQUE, AUTOMATIQUE** **https://edeea.universite-lyon.fr** Sec. : Stéphanie CAUVIN Bâtiment Direction INSA Lyon Tél : 04.72.43.71.70 secretariat.edeea@insa-lyon.fr	**M. Philippe DELACHARTRE** INSA LYON Laboratoire CREATIS Bâtiment Blaise Pascal, 7 avenue Jean Capelle 69621 Villeurbanne CEDEX Tél : 04.72.43.88.63 **philippe.delachartre@insa-lyon.fr**
E2M2	**ÉVOLUTION, ÉCOSYSTÈME, MICROBIOLOGIE, MODÉLISATION** **http://e2m2.universite-lyon.fr** Sec. : Bénédicte LANZA Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 secretariat.e2m2@univ-lyon1.fr	**Mme Sandrine CHARLES** Université Claude Bernard Lyon 1 UFR Biosciences Bâtiment Mendel 43, boulevard du 11 Novembre 1918 69622 Villeurbanne CEDEX **sandrine.charles@univ-lyon1.fr**
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTÉ** **http://ediss.universite-lyon.fr** Sec. : Bénédicte LANZA Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 secretariat.ediss@univ-lyon1.fr	**Mme Sylvie RICARD-BLUM** Institut de Chimie et Biochimie Moléculaires et Supramoléculaires (ICBMS) - UMR 5246 CNRS - Université Lyon 1 Bâtiment Raulin - 2ème étage Nord 43 Boulevard du 11 novembre 1918 69622 Villeurbanne Cedex Tél : +33(0)4 72 44 82 32 **sylvie.ricard-blum@univ-lyon1.fr**
INFOMATHS	**INFORMATIQUE ET MATHÉMATIQUES** **http://edinfomaths.universite-lyon.fr** Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage Tél : 04.72.43.80.46 infomaths@univ-lyon1.fr	**M. Hamamache KHEDDOUCI** Université Claude Bernard Lyon 1 Bât. Nautibus 43, Boulevard du 11 novembre 1918 69 622 Villeurbanne Cedex France Tél : 04.72.44.83.69 **hamamache.kheddouci@univ-lyon1.fr**
Matériaux	**MATÉRIAUX DE LYON** **http://ed34.universite-lyon.fr** Sec. : Yann DE ORDENANA Tél : 04.72.18.62.44 yann.de-ordenana@ec-lyon.fr	**M. Stéphane BENAYOUN** Ecole Centrale de Lyon Laboratoire LTDS 36 avenue Guy de Collongue 69134 Ecully CEDEX Tél : 04.72.18.64.37 **stephane.benayoun@ec-lyon.fr**
MEGA	**MÉCANIQUE, ÉNERGÉTIQUE, GÉNIE CIVIL, ACOUSTIQUE** **http://edmega.universite-lyon.fr** Sec. : Stéphanie CAUVIN Tél : 04.72.43.71.70 Bâtiment Direction INSA Lyon mega@insa-lyon.fr	**M. Jocelyn BONJOUR** INSA Lyon Laboratoire CETHIL Bâtiment Sadi-Carnot 9, rue de la Physique 69621 Villeurbanne CEDEX **jocelyn.bonjour@insa-lyon.fr**
ScSo	**ScSo*** **https://edsciencessociales.universite-lyon.fr** Sec. : Mélina FAVETON INSA : J.Y. TOUSSAINT Tél : 04.78.69.77.79 melina.faveton@univ-lyon2.fr	**M. Bruno MILLY** Université Lumière Lyon 2 86 Rue Pasteur 69365 Lyon CEDEX 07 **bruno.milly@univ-lyon2.fr**

*ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

PhD Thesis

Navigation Among Movable Obstacles (NAMO) Extended to Social and Multi-Robot Constraints

Presented before
L'Institut National des Sciences Appliquées de Lyon

For
The grade of Doctor

By
Benoit Renault

Doctoral School
Infomaths

Advisors
Olivier Simonin
Jacques Saraydaryan

Thesis Committee
Julie Dugdale (President)
Philippe Mathieu (Reviewer)
Fabien Michel (Reviewer)
Rachid Alami (Examiner)

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Defended on December 19th, 2023

Abstract

Navigation Among Movable Obstacles (NAMO) Extended to Social and Multi-Robot Constraints

Benoit RENAULT

As robots become ever more commonplace in human environments, taking care of ever more tasks such as cleaning, security or food service, their current limitations only become more apparent. One such limitation is of their navigation capability in the presence of obstacles: they always avoid them, and freeze in place when avoidance is impossible.

This is what brought about the creation of Navigation Among Movable Obstacles (NAMO) algorithms, expected to allow robots to manipulate obstacles as to facilitate their own movement. However, these algorithms were designed under the hypothesis of a single robot per environment, biasing NAMO algorithms into only optimizing the single robot's displacement cost - without any consideration for humans or other robots. While it is desirable to endow robots with the human capability of moving obstacles, they must however do so while respecting social norms and rules of humans.

We have thus extended the NAMO problem as to take into account these new social and multi-robots aspects. By relying on the concept of affordance spaces, we have developed a social occupation cost model allowing the evaluation of the impact of moved objects on the environment's navigability. We implemented (and improved) reference NAMO algorithms, in our open source simulation tool, and modified them so that they may plan compromises between robot displacement cost and social occupation cost of moved obstacles - resulting in improved navigability. We also developed an implicit coordination strategy allowing the concurrent execution of these same algorithms by multiple robots as is, without any explicit communication requirements, while preserving the no-collision guarantee; verifying the relevance of our social occupation cost model in the actual presence of other robots. As such, this work constitutes the first steps towards a Social and Multi-Robot NAMO.

Acknowledgements

I am grateful to my advisors, Olivier Simonin and Jacques Saraydaryan, for their unwavering support and guidance throughout the five years of this thesis. Their balanced input and continuous encouragements have definitely brought me further on my path as a scientist, and as a person. It is firstly thanks to them that I could pursue this thesis, and finally bring it to fruition, despite the many obstacles they have helped me push through.

I also thank my colleagues from the CITI Laboratory and Chroma Team, for all the good moments and discussions we shared together. You all have made this travel in the world of research that much more passionating. These thanks also extend to all INSA Lyon personnel that have accompanied me in the ten years I have spent on the Doua Campus.

I would also like to express my gratefulness to the teachers, professors and mentors who guided me throughout my youth, for teaching me - and all of their students- to strive for truth, understanding, and knowledge. Their impact is far greater than any of us could imagine.

My final thanks go to my friends and family, without whom I simply would not have gone this far. Thank you for being there for me, during good or bad times, and for bringing joy in my life. I shall always strive to return the same faithfulness and joy. It is a privilege to have all of you in my existence, and I am forever thankful to all of you.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

AABBTree	**A**xis-**A**ligned **B**ounding **B**ox **T**ree
BA	**B**locking **A**reas
BHPN	**B**elief **H**ierarchical **P**lanner in the **N**ow
B(est)-FS	**B**est-**F**irst **S**earch
B(readth)-FS	**B**readth-**F**irst **S**earch
CBS	**C**onflict-**B**ased **S**earch
CITO	**C**ontact-**I**mplicit **T**rajectory **O**ptimization
C-NAMO	**C**oordinated **N**avigation **A**mong **M**ovable **O**bstacles
CSV	**C**onvex **S**wept **V**olume
DFS	**D**epth-**F**irst **S**earch
DOF	**D**egree(s) **O**f **F**reedom
$\mathcal{LP}$	**L**inear Rearrangement Planning **P**roblem class of Ben Shahar & Rivlin [24]
$L(P)_n$	**L**inear NAMO (**P**roblem) of class n
$L(P)_nM$	**L**inear and **M**onotone NAMO (**P**roblem) of class n
GDAL	**G**eospatial **D**ata **A**bstraction **L**ibrary
GEOS	**G**eometry **E**ngine, **O**pen **S**ource
HRI	**H**uman-**R**obot **I**nteraction
KDRRT	**K**ino-**D**ynamic **R**apidly **E**xploring **R**andom **T**ree
KPIECE	**K**inodynamic **P**lanning by **I**nterior-**E**xterior **C**ell **E**xploration
MAPF	**M**ulti-**A**gent **P**ath **F**inding
MCMC	**M**arkov **C**hain **M**onte **C**arlo
MCTS	**M**onte **C**arlo **T**ree **S**earch
MDP	**M**arkov **D**ecision **P**rocess
MRCR	**M**ulti-**R**obot **C**lutter **R**emoval
MR-NAMO	**M**ulti-**R**obot **N**avigation **A**mong **M**ovable **O**bstacles
MRS	**M**ulti-**R**obot **S**ystem
MuJoCo	**Mu**lti-**Jo**int dynamics with **Co**ntact (physics engine)
NAMO	**N**avigation **A**mong **M**ovable **O**bstacles
NL	**N**on-**L**inear NAMO problem
NM	**N**on-**M**onotone NAMO problem
NP-hard(ness)	**N**on-deterministic **P**olynomial-time **hard(ness)**
OInP	**O**bject **In** **P**ath potential conflict type
PedSim	**Ped**estrian **Sim**ulator
PLR	**P**ath of **L**east **R**esistance

POMDP	**P**artially-**O**bservable **M**arkov **D**ecision **P**rocess
PRM	**P**robabilistic **R**oad-**M**ap
PYPL	**P**opularit**Y** of **P**rogramming **L**anguage Index
QGIS	**Q**uantum **G**eographic **I**nformation **S**ystem
RGB-D	**R**ed **G**reen **B**lue - **D**epth
ROS	**R**obot **O**perating **S**ystem
R-R	**R**obot-**R**obot potential conflict type
RRT	**R**apidly-exploring **R**andom **T**ree
SAN	**S**ocially-**A**ware **N**avigation
SC-NAMO	**S**ocial **C**oordinated **N**avigation **A**mong **M**ovable **O**bstacles
SCVX	**S**uccessive **C**on**V**e**X**ification
SGrab	**S**imultaneous **G**rab potential conflict type
SingO	**S**teal**ing** **O**bject potential conflict type
S-NAMO	**S**ocial **N**avigation **A**mong **M**ovable **O**bstacles
SO	**S**tolen **O**bject potential conflict type
SPC	**S**ocial **P**lacement **C**hoice NAMO Problem
SSA	**S**imultaneous **S**pace **A**ccess potential conflict type
T(A)MP	Integrated **T**ask (**A**nd) **M**otion **P**lanning
VG	**V**isibility **G**raph

Dedicated to my parents.

Chapter I

Introduction

I. 1 Motivation

Service robotics are steadily becoming more common within households, with almost 18.5 million new consumer service robots being sold in 2020 (+6% compared to 2019), with most notably autonomous cleaning robots representing more than 92% of these numbers (17.2 million units) [25]. As robots become more commonplace, their current limits get more visible: one such limit is the matter of how robots deal with movable obstacles when navigating. By that, we mean any object that does not define the very layout of the environment and is thus meant to be frequently manipulated by humans: chairs, bags, boxes, or smaller clutter such as toys, plushes, etc. by opposition to walls, tables, shelves, and any other piece of heavy furniture that isn't fitted with wheels. Upon reading this, you may have immediately pictured a messy bedroom in your mind (more likely so if you have lively children at home), such as the one in Fig.I.1. Current household robots only know to avoid such clutter - when they do detect it (they otherwise hazardously run through it). In the case presented in Fig.I.1, avoidance is arguably safest, it would also mean failure of the cleaning mission altogether, since the robot would simply never enter the space to be cleaned; and mindlessly going through would damage the robot or objects.

FIGURE I.1: Example of a cluttered space hampering a cleaning robot's task
(Credit: Nate Smith - https://youtu.be/cXdg3OO-2tE)

One can easily extrapolate this situation to any navigation scenario in other human environments such as offices, storage rooms, or even industrial environments such as workshops. In any case, carefully moving the blocking objects aside and resuming navigation rather than failing whatever mission the robot is required to accomplish would be considered a better outcome. That is why even major robotics companies consider it to be one of the necessary next advancements for household robots [26].

Similar considerations are what led Stilman et al. [3], in 2005, to the formulation of the **Navigation Among Movable Obstacles (NAMO)** problem: computing *a single robot*'s collision-free plan from a start to a goal configuration, allowing the manipulation of movable obstacles to minimize a displacement cost function (generally expressed as an estimate of traversed distance, elapsed time, or expended energy), as illustrated in Fig.I.2. This problem extends the fundamental "Piano Mover" problem, only adding this manipulation capability. Since then, in order to make and reliably use NAMO plans in the real world, the study of NAMO has most notably been expanded to account for state/action uncertainties and realistic kinematic and dynamic considerations, with the work of Levihn and Scholz [27, 28].

(A) Start configuration (B) After 45s (C) After 180s

FIGURE I.2: A real-world NAMO experiment in a controlled lab environment, showcasing the results of Stilman's research. The robot autonomously identifies the surrounding objects, and determines that it needs to move the chair to reach its goal, and does so.

However, until now, to the best of our knowledge, has never been considered in NAMO problems the presence of more agents (i.e. humans and robots) than the robot executing the NAMO plan itself. That is the case although agents are the main reason clutter appears[1]; and robotics applications imply ever more robots operating in a shared environment, even for home cleaning scenarios [29]. As a consequence, existing NAMO approaches only optimize the displacement cost of the robot and thus tend to move obstacles to the closest available configuration that allows them to pass [30]. This can result in disputable behavior like in Fig.I.3: the robot picks up a toy and releases it behind itself, as to free its path forward, but blocking the way again for any other agent that may come after - or even its future self! While it did help the robot achieve its required goal, robots should arguably execute NAMO plans without causing such trouble to humans or robots.

It is this lack of study of NAMO algorithms' effects in human environments or in the presence of other agents that primarily motivates our present work. The concrete observation of

[1]beyond catastrophic events such as a shelf breaking or an earthquake shaking

such self-centered behavior of existing NAMO algorithms, and its evident inadequacy in human environments meant to be shared, further affirm the need to explore the social and multi-agent implications of NAMO.

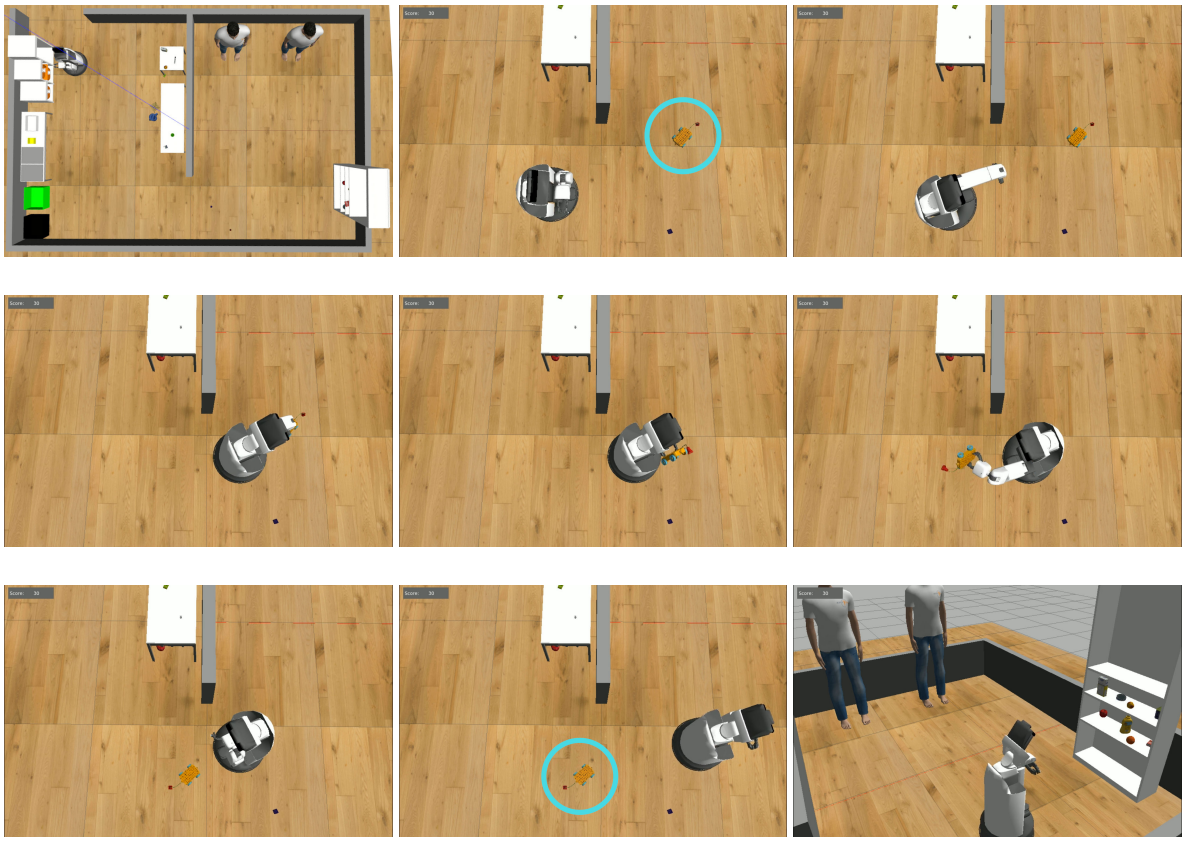


FIGURE I.3: Virtual home scenario snapshots from our Robocup@Home 2021 competition participation. The robot had to execute a household task in each room, but small objects would hamper passage in-between, and were thought to be avoided. We however interpreted and solved this as a NAMO problem, whereas other teams would simply be blocked due to the difficulty of circumventing the objects without touching them involuntarily. Our performance was rewarded with a specific "Best Cleanup Award", in addition to our placing second in our league.

I. 2 Challenges

As we have just discussed, executing NAMO algorithms in human environments or in the presence of other agents presents non-trivial open challenges. Among these, we will not discuss *sensing&action uncertainty* nor *kinodynamic physical interactions*, as they have already been well-covered in previous already cited literature [27, 28]. It makes more sense to try and address the specific challenges brought by our new focus on human environments and agent plurality:

Social Acceptability One can not simply introduce robots in human living spaces expecting everything to work out from the get-go. Humans are social beings, relying on social norms and

rules to interact with one another, be it directly or through their environment. It is a widely-recognized fact that robots are expected to follow these same norms and rules, if they are to ever be welcomed in human spaces [13]. Finding, modeling and having robots follow them is in itself a huge field in robotics research [31] - and we will need to study how this applies to our NAMO context of navigating and manipulating obstacles, especially when it comes to their placement in the environment.

Coordination Multiple robots planning and concurrently executing plans in a shared environment requires coordination in order to guarantee the absence of collisions, as clearly stated in the overwhelming literature on the matter [16]. It opens a myriad of questions as to how robots may or not interact with one another: do they communicate ? How ? What information do they share, if at all ? Can they manipulate a same obstacle together ? How to predict the behavior of another robot if it cannot explicitly communicate what it wants to do ? ... We must thus study these implications in our NAMO context too.

Dimensionality The most fundamental challenge of both NAMO and Multi-Robot coordination is the curse of dimensionality. To better understand what we mean by this, let us consider the complexity of planning for multiple agents (robots) among multiple movable obstacles. The full configuration space[2] for M robots and N obstacles respectively with d_r and d_o degrees of freedom is the product of each individual robot and obstacle's configuration space:

$$\mathcal{C}_{r_1} \times \mathcal{C}_{r_2} \times ... \times \mathcal{C}_{r_M} \times \mathcal{C}_{o_1} \times \mathcal{C}_{o_2} \times ... \times \mathcal{C}_{o_N} \in \mathbb{R}^{Md_r+Nd_o}$$

In a discretized search space of resolution r for each dimension, this yields a number of configurations exponential in the number of obstacles **and** robots, asymptotically noted as $\mathcal{O}(r^{Md_r+Nd_o})$. In a simple square grid of side 10 cells, with just 2 robots and 2 obstacles able to move in both axes, this would already mean a hundred million configurations to search through, obviously making brute-force search unthinkable. To solve such problems, with the additional requirements of respecting social norms and rules, we will need to carefully consider computation optimization strategies and heuristics.

I. 3 Approach

By now, the very title of this thesis and the previous sections should have made rather clear which direction, or bias, we choose to study this problem of having multiple agents navigate in a shared human environment: we extend existing NAMO research so that it can operate in this new context. This comes with the main benefit of already partly addressing the aforementioned *Dimensionality* challenge, thanks to it being profusely addressed in the existing NAMO literature. This approach makes sense, since the application scenarios we are aiming for contain far more movable obstacles than robots.

[2]The configuration of a robot/object can be understood as its coordinates in euclidean space. The configuration space refers to the mathematical space of all possible configurations of all the robots and obstacles.

We thus first explore said NAMO literature in most detail, in order to select relevant algorithms to serve as base for extensions that will address the *Social Acceptability* and *Coordination* challenges. They also serve as a baseline to compare against their extended versions in our experiments, through our open source ad-hoc NAMO simulation tool and datasets.

To specifically address the *Social Acceptability* challenge, we rely on the social navigation concept of Affordance Spaces [32], and first propose a naive social model of "taboo zones", or forbidden affordance spaces. We then propose a second more general and scalable *Social Occupation Cost* model representing an affordance space of *accessibility/navigability* based on two heuristic hypotheses: avoiding narrow spaces, and not leaving objects in the middle of space. We show how to modify existing NAMO algorithms to use these models to affect the placement of manipulated obstacles, as to help avoid non socially-acceptable environment modifications such as previously shown in Fig.I.3.

To address the challenge of *Coordination*, we propose an implicit coordination algorithm to allow concurrent use of existing NAMO algorithms by multiple robots, while still guaranteeing the absence of collisions - without any explicit communication requirement, ensuring maximum robustness regarding the possibility of individual robot or communication failure. We also show how our *Social Occupation Cost* model can help resolve deadlocks in this no-explicit communication allowed context - and also improve navigation for all robots in the environment.

I. 4 Thesis overview

I. 4.1 Summary of Contributions

The contributions of this thesis are summarized as follows, in order of appearance:

- **An extensive survey of NAMO literature** — the first such survey, chronologically covering NAMO from its roots and inception, to its various extensions and applications.
- **Improvements of existing reference NAMO Algorithms** — namely (Stilman, 2005)[3, 19] and (Wu&Levihn, 2014)'s [33, 34, 35, 7] algorithms. We define a common formalism to serve this rewrite and provide a basis for our problem extensions.
- **Open simulator and data** — development of an open simulation tool with implementations of the aforementioned algorithms and their extensions discussed below, with open experiment input and output data.
- **S-NAMO** — the general Socially-aware NAMO problem, extending NAMO with social constraints, to also allow a robot to reason about its decisions' impact on the environment's accessibility for humans.
- **Social Placement Cost Model** — definition of a generic model of the accessibility/navigability affordance offered by free space in human environments, and as a corollary, of the disturbance to humans caused by movable obstacles. This low-requirement model solely relies on the analysis of the environment's binary occupancy grid of fixed obstacles.

- **S-NAMO Algorithm** — a focused variant of (Stilman, 2005)'s NAMO Algorithm towards the best compromise solution between robot displacement cost and social occupation cost discussed above.
- **MR-NAMO** — definition of the general Multi-Robot NAMO problem, generalizing the NAMO domain to also allow a robot to reason about other robots.
- **C-NAMO** — an implicit coordination strategy that can leverage existing NAMO algorithms as is, without requiring explicit communication between robots, to provide solutions to MR-NAMO problems, including deadlock situations.

I. 4.2 Document Outline

This thesis is structured as follows:

- **Chapter II - State of the Art** provides an in-depth survey of existing NAMO literature, as well as an overview of Social Navigation and Multi-Robot Coordination problems with a discussion of their relation to NAMO.
- **Chapter III - Revising and simulating reference NAMO algorithms** provides a common formalism for the two baseline NAMO algorithms of (Stilman, 2005) and (Wu&Levihn, 2014) we build upon in the following chapters, revisions of said algorithms, and a description of their implementation in our open simulator and data, as well as a presentation of this simulator we developed.
- **Chapter IV - Socially-aware NAMO** introduces the novel problem of Socially-Aware NAMO (S-NAMO), our new Social Placement Cost Model and S-NAMO algorithm that makes use of it, as well as experiments in increasingly complex environments to measure its impact on the environment's accessibility for humans.
- **Chapter V - Multi-Robot NAMO** introduces the new problem of Multi-Robot NAMO (MR-NAMO), and the local coordination approach we devised to solve MR-NAMO problems, including deadlocks. We conclude by measuring the synergy between our S-NAMO and MR-NAMO contributions in another set of experiments.
- **Chapter VI - Conclusion** provides a synthesis of this work and outlines research perspectives from there.

Chapter II

State of the Art

In this chapter, we first provide a chronological and detailed overview of the NAMO literature, as it is the core of our thesis work. From even before the definition of the actual NAMO problem, to its latest applications, we discuss how NAMO has been addressed and extended through the years, along with relevant applicable vocabulary. This extensive overview allows us to conclude as to the characteristics and limitations of the existing body of work, with the noteworthy conclusion that Social and Multi-Robot considerations have yet to be applied to the NAMO problem. As we present the NAMO literature, we also introduce the most-closely related problems to NAMO: Navigation Planning, Manipulation Planning, Rearrangement Planning, Assembly Planning, and the overarching domain of combined Task and Motion Planning. In the second section, we provide an overview of existing Social and Multi-Robot extensions to these similar problems, and introduce relevant associated concepts. These concepts help us clarify our vision of the Social and Multi-Robot NAMO problem we address in this thesis, and shall be used in our own algorithmic extensions in the following chapters. The present chapter is meant to be read linearly, as one would a story: hence, for the sake of simplicity and understanding, the use of mathematical notations, formulas and abbreviations is voluntarily kept to a minimum.

II. 1 Navigation Among Movable Obstacles (NAMO):

II. 1.1 "Motion planning in the presence of movable obstacles"

In 1988, 17 years before Stilman coined the term of NAMO, Gordon Wilfong was the first to introduce the problem of "finding collision-free motions in a changeable workspace" [36, 37][1] - that is, to compute the motion of a robot to a goal position that is allowed to move obstacles (their final positions being or not part of the goal). Their paper demonstrated that, what Wilfong then called "the movable-obstacle problem", was PSPACE-hard when the final positions of the obstacles were specified (i.e. Rearrangement Planning), and otherwise **NP-hard (i.e. NAMO)** - Fig.II.2 provides a reminder of the various complexity classes in computational complexity theory. This demonstration was done using a simplified world representation of

[1] Several approaches/algorithms presented in this NAMO literature overview have been published in different conferences/journals/thesis documents, with slightly variable content such as new figures or explanations. For each such instance, we first cite all publications, then refer to the algorithm with the reference we have found to be most relevant among these, for the sake of readability.

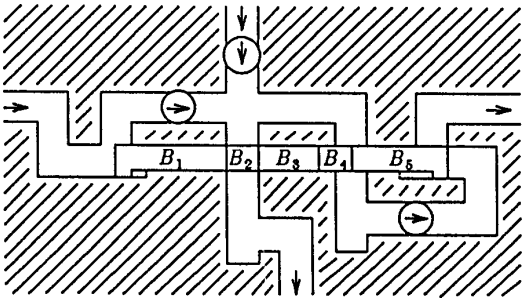


FIGURE II.1: A NAMO problem used by Wilfong in their NP-hardness demonstration. Static obstacles are dashed, movable ones are annotated with a B. Several cases are studied, hence why the robot start can be either of the arrowed circles, and goal either of the arrows.

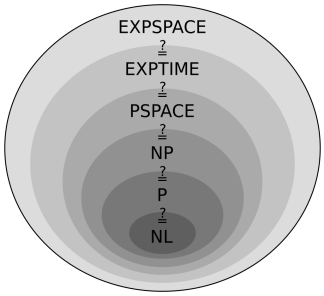


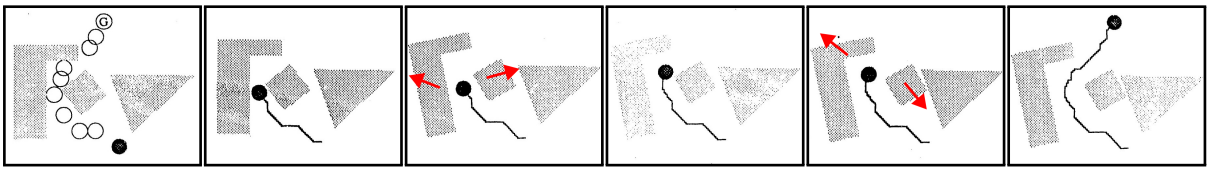
FIGURE II.2: A representation of complexity classes - from the Wikipedia article on Computational complexity theory

the problem (without loss of generality), where the robot is considered as a square, all planar obstacles as rectangles of four sizes or "L-Shaped", parallel to the x- or y-axis, as illustrated in Fig.II.1. Crudely put, this demonstration showed that complete planning for the full NAMO domain is computationally infeasible.

The same paper included a resolution algorithm for the specific case of the manipulation of a single convex polygonal movable obstacle blocking the way of a convex polygonal robot, in a world composed of not necessarily convex polygons. Robot motion was restricted to translations, and grasping the obstacle was assumed possible only when the robot entered in edge contact with the obstacle (in their words: "share a common wall") - common interaction hypotheses in subsequent papers, as we shall see in the next paragraphs. To the best of our knowledge, this algorithm has however not been implemented nor experimentally verified, as the author did not provide any resolved scenario with it, nor experimental results. This, **with the algorithm's fundamental limitation to single-obstacle problems**, explain why **it cannot be considered as the first NAMO algorithm in the literature**.

II. 1.2 The first NAMO planning algorithm

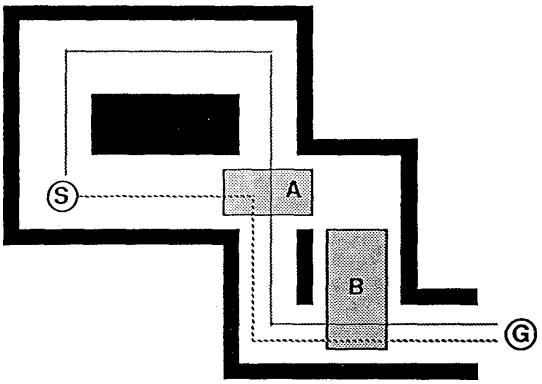
In 1991, Chen and Hwang introduced the **first planner with the ability to manipulate multiple movable obstacles** to reach a goal robot configuration [1, 38]. This planner was designed to work for a circular robot in an environment where obstacles are represented by unions of convex polygons (e.g. a concave L-shape obstacle such as B_1 or B_5 in Fig.II.1 would be represented by the union of two rectangles), and associated with a positive mass value, used to estimate the required work to move them. Both the robot and obstacles may be translated or rotated, and the robot is supposed to possess an invisible arm capable of moving any obstacle in any direction, on the condition that the robot gets in contact with the object first (Cf. Fig.II.3b). In contrast with the later NAMO literature, the planner accounts for movement transfer from the manipulated object to other objects, allowing the robot to "plow" through obstacles (i.e. using one obstacle to push others).



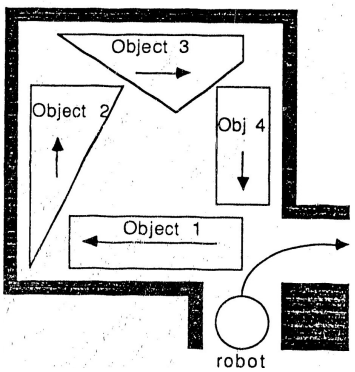
(A) (B) (C) (D) (E) (F)

FIGURE II.3: Example NAMO problem resolution by Chen&Hwang's algorithm [1]. Robot is black circle, goal is white circle with a "G", movable obstacles are polygons. In (A), the global planner path is shown as white circles. The local planner verification sequence is shown in (B) to (F). In (B), the robot first "slides" to the configuration neighboring the square and L-shaped obstacles, then "shoves aside" both obstacles in (C), to clear the path to the next configuration shown in (D), where they are "shoved aside" again in (E) so that the robot may "slide" to the goal in (F).

Their planner can be summarized as a search for a Path of Least Resistance (abb. PLR). A **global** planner randomly samples robot configurations in a grid decomposition of the space (circles in Fig.II.3a), then computes heuristic work underestimates of clearing obstacles from these configurations. The global planner then searches the graph of neighboring configurations using Dijkstra's algorithm to find a path of least cost, that may pass through obstacles. This path is verified by a **local** planner, that applies primitive operators to connect two neighboring configurations. These operators either "slide" the robot from a free configuration to another, or "shove aside" obstructing objects to clear the heuristic path (that is, use its invisible arm to move the objects)[2].



(A) To clear the shortest path (dashed line) computed by the global planner, the robot "shoves aside" Object A right, blocking object B. A different algorithm is needed to find the solution sequence: (A, left), (B, top) using a detour by the top (solid line).



(B) To clear the shortest path (curved arrow) computed by the global planner, the robot "shoves aside" Object 1 left, resulting in Object 4 blocking the path in reaction. A different algorithm is needed to find the solution sequence: (1, right), (4, bottom), (3, right), (2, top), (1, left).

FIGURE II.4: Unsolvable NAMO cases for Chen&Hwang's algorithm [1].

While this heuristic approach is effective on some examples like Fig.II.3, it suffers from a lack of backtracking - that is, the algorithm only considers a single path per reachable robot configuration. In Fig.II.4a, this prevents the computation of a detour around the top (solid line),

[2]When failing to clear obstacles from a configuration after calling these operators a fixed maximum number of times, the configurations graph and the Dijkstra path search tree are dynamically updated to remove said configuration. The search is then resumed from the last valid configuration.

after first considering the shortest path (dashed line). This results in algorithmic **incompleteness** (no guarantee to find a solution if one exists), and the authors suggest that **a symbolic task-level planner would be required to solve such cases**. Finally, the local nature of obstacle manipulations to join neighboring robot configurations, and the hypothesis of movement transfer from the manipulated object to others, both limit the robot's planning capabilities to "plowing" paths. That is, paths where all obstacles are greedily pushed away from the robot as it moves forward, without proper consideration for the (positive or negative) effects it may have in distant portions of the environment (as illustrated in both Fig.II.4a and II.4b, where the robot's focus on the first accessible obstacle prevents it from considering the movement of other obstacles that should have been moved first).

II. 1.3 Task-level planning for LP1 NAMO Problems

The movable-obstacle problem gained renewed interest in 2004, with the publication of two new planners by Okada [2] and Stilman [39, 3, 19], each with their own **task-level hierarchization**. Both were inspired by Alami et al.'s task-level decomposition strategy for manipulation planning [40, 41], consisting in building a sequence of transit (navigation) and transfer (manipulation) paths, where the robot respectively navigates alone or while holding an obstacle, separated by rigid grasp and ungrasp operations.

Assuming a fully known simulated 3D world model with predefined grasping points and associated manipulation methods (Cf. Fig.II.5.A), Okada et al. [2] chose to build a task graph (Cf. Fig.II.5.B) where nodes are movable obstacles, and edges represent lowest-cost navigation paths between valid robot standing configurations near movables obstacles that allow grasping (Cf. Fig.II.5.B.5). While the exact navigation planning algorithm (Cf. Fig.II.5.B.3) is not specified beyond its 3-dimensional search space - x, y, θ, it is stated that the manipulation planner (Cf. Fig.II.5.B.4) uses a Rapidly-exploring Random Tree (abb. RRT) that samples configurations over the full body joint space, assuming the obstacle is fully constrained by the robot's grasp. The RRT manipulation search is given a final obstacle configuration as goal, which is obtained by sampling candidate obstacle configurations in a grid space decomposition (Cf. Fig.II.5.B.6.Right), until one is found that does not intersect with a heuristic navigation path from the start to the goal location ignoring the movable obstacle (Cf. Fig.II.5.B.5.Left). The manipulation planner yields a work cost estimate for the task graph nodes, and the full motion plan is computed through a "standard graph search" algorithm (most likely Dijkstra's), using either this work estimate (Cf. Fig.II.5.B.1) **or** the navigation planner's distance cost estimate (Cf. Fig.II.5.B.2). While seemingly actionable at least in simulation, as the figure shows, *this approach is however not provably complete nor optimal*. Also, it is very computationally inefficient, as **it requires the computation of the full task graph** - thus also the computation of all navigation and manipulation sequences, even for objects that are very unlikely to open a path to the goal.

In parallel, Stilman and Kuffner [39, 3, 19] first introduced a formal definition for the now widely used term of Navigation Among Movable Obstacles (abb. NAMO). As for their task-level decomposition, they went with a somewhat opposite task graph structure, where edges are movable obstacles and nodes are *components of the robot's free configuration space* (syn. *free*

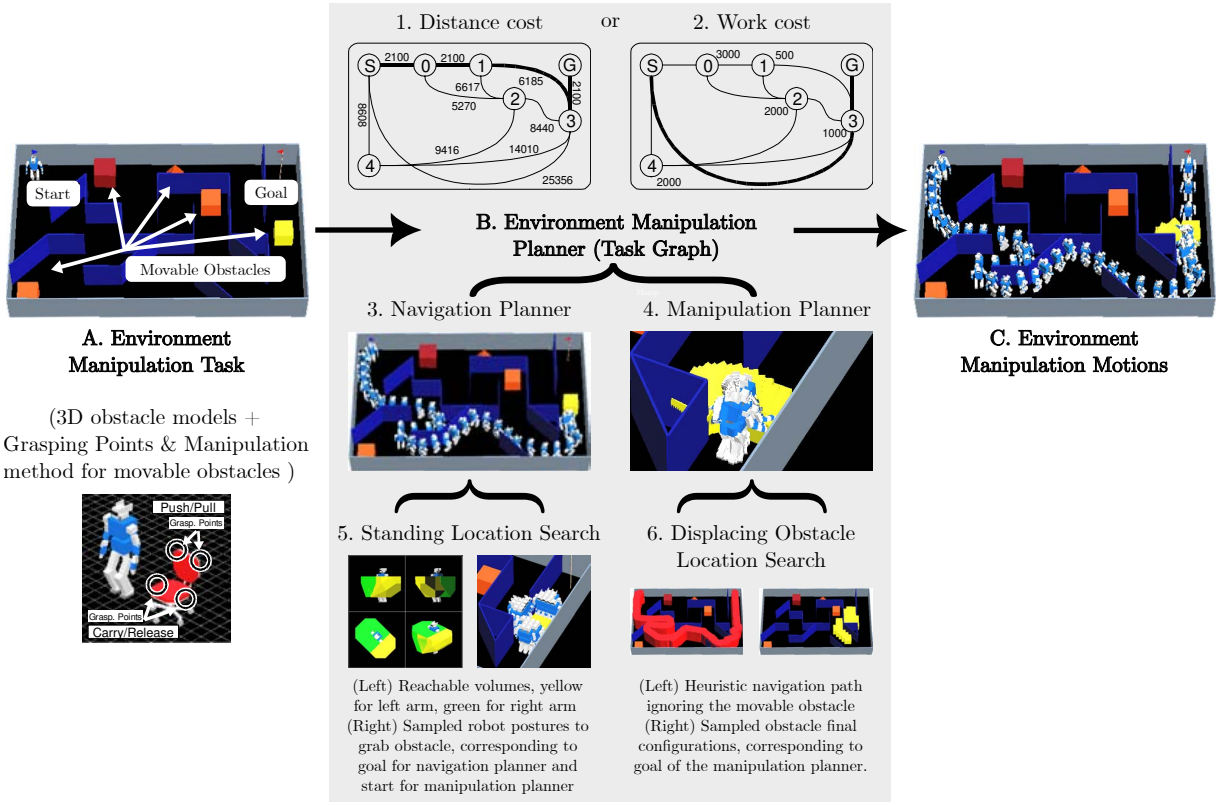


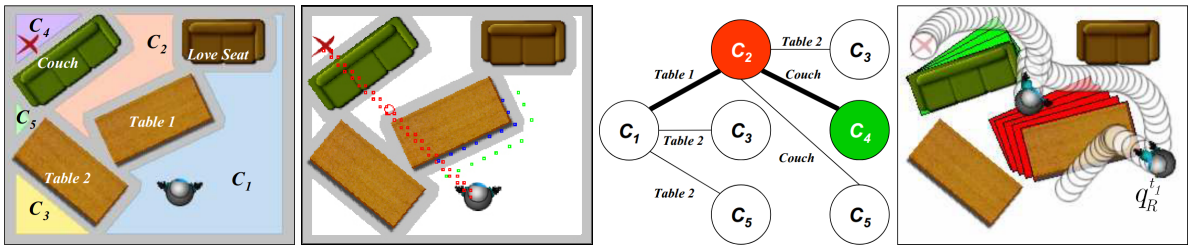
FIGURE II.5: Example NAMO problem resolution by Okada et al.'s algorithm [2]. On the left, the world model description, in the middle, the planner and its subcomponents, on the right, the planned motion sequence to solve the problem.

space components). Simply put, a free space component can be understood as an area where the robot may navigate from any point to another without having to manipulate any obstacle, as illustrated in Fig.II.6a. It is actually this task graph model that allowed them to devise a fundamental subclass of NAMO problems called Linear Problems (abb. LP), by analogy to rearrangement planning [24]. By their definition, a problem belongs to the LP class, if it can be decomposed into a sequence of **independent sub-problems**, that each consist in connecting two components of free space by moving one or more objects (such a sub-problem being called a "**keyhole**"). Here, "Independent" means that any action used in solving a keyhole does not interfere with the actions needed to solve any subsequent keyhole. Correspondingly, LP1 (also written L_1), refers to the class of problems for which disconnected components of free-space can be connected independently by moving a single obstacle, as is the case with the example of Fig.II.6 we will be discussing. Conversely, by definition, Non-linear NAMO problems (abb. *NL*) cannot be decomposed in a sequence of keyholes that can be solved independently[3].

Stilman & Kuffner made use of this task-level decomposition to devise (but not implement) a theoretical resolution-optimal planning algorithm, and implement an efficient resolution-complete (but not optimal) algorithm, for this class of LP1 problems[4]. Stilman & Kuffner's

[3]One example of such a problem is provided later in Fig.II.10a.

[4]The term "resolution"-optimal/complete is used here to underline the use of purely graph-based navigation and manipulation search methods (respectively A^* and Breadth-First Search), on a 2D grid graph with a specific square side-length *resolution*. It also refers to the fixed number, for each obstacle, of grasping points and corresponding robot grasping configurations (grid cells) from which they can be reached (Cf. Fig.II.6b).



(A) Initial world state with color-filled free space components $\{C_1, C_2, C_3, C_4, C_5\}$

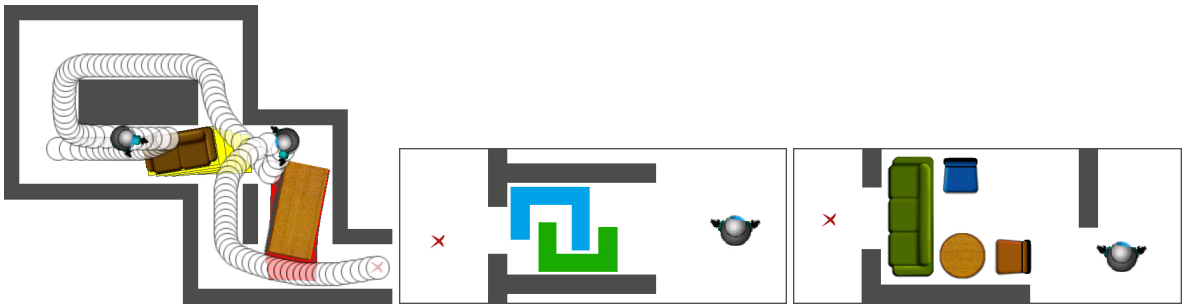
(B) Blue dots are grasping points, green dots are corresponding robot configurations. Red dots are the heuristic path traversing obstacles.

(C) Task-level Best-First Search tree, with obstacles as edges and components as nodes

(D) Solution motion sequence yielded by Stilman&Kuffner's algorithm [3]

FIGURE II.6: Example LP1 NAMO problem resolution by Stilman&Kuffner's algorithm [3]. The robot navigates in C_1 to transfer the *Table 1* to fuse C_1 with C_2, then navigates to the Couch to fuse C_2 with C_4 in order to finally navigate to its goal.

resolution-optimal theoretical algorithm is structurally similar to Okada et al.'s algorithm [2], making it no more computationally efficient because of the exact same requirement of computing the full task graph. That is why they rather devised and implemented a resolution-complete greedy variant, that instead of the full task graph, recursively computes a Best-First Search task tree (Cf. Fig.II.6c). In order to select the "best" obstacle/component pair to connect, they used an additional relaxed-constraint navigation planner based on the A^* algorithm, that was allowed to pass through movable obstacles for an additional heuristic traversal cost (Cf. Fig.II.6b)[5]. This allowed them to compute solutions within seconds, such as the one in Fig.II.6d, or to the previously unsolvable problem for Chen&Hwang's algorithm [1] (Cf. Fig.II.7a).



(A) Unsolvable L_1 problem for Chen&Hwang's algorithm [1] (Cf. Fig.II.4a), solved by Stilman&Kuffner's algorithm [3]

(B) Strongly coupled L_2 problem - The two obstacles need to be repeatedly and alternatively manipulated, or grabbed together

(C) L_4 problem - The couch's movement is constrained by the table's, which is also constrained by the chairs'.

FIGURE II.7: Capabilities and limitations of Stilman&Kuffner's algorithm [3]. (A) is a previously unsolved problem resolved by Stilman&Kuffner's algorithm, while (B) and (C) cannot be solved by it (and most likely neither Okada et al.'s algorithm [2]).

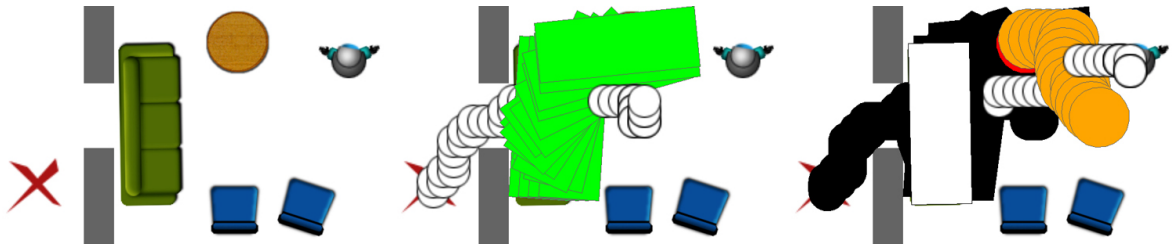
Both Okada et al. [42] and Stilman & Kuffner [43, 44, 19] eventually bridged the gap to a real-world experiment with an HRP-2 humanoid robot , but where the robot only moves a

[5]We provide an in-depth explanation and revisited pseudocode formalization in the next chapter, Section III. 2.2.

single obstacle over a very short distance to minimize action uncertainty, and robot sensing capabilities are augmented with external sensors. While these experiments make them both actionable algorithms for solving real-world NAMO problems, the completeness guarantee and the capacity of Stilman & Kuffner's algorithm to larger scale environments of experimentally up to 90 obstacles, make it an overall better NAMO algorithm[6]. In any case, neither of these algorithms could solve the problems illustrated in Fig.II.7 [B, C], as they are not of L_1 class.

II. 1.4 Reverse-search planning for L_kM NAMO Problems

Actually, the problem in Fig.II.7c could be solved by moving each blocking obstacle only once - if only they were moved in the right order. Inspired by the literature of Assembly Planning[7] [45], Stilman & Kuffner described such problems as Monotone (abb. M) [46, 47, 19]. Conversely, they defined situations like in Fig.II.7b as Non-Monotone (abb. NM_i) problems, that need to be expressed as i monotone problems with intermediate states to be solved. In the linear and monotone problem illustrated in Fig.II.7c, it clearly appears that first reasoning about the obstacles closest to the goal (the couch), allows to then choose placements for other obstacles that would not interfere with the couch's displacement. It is this consideration that lead Stilman & Kuffner to the formulation of a new *reverse-planning* algorithm for solving *Monotone Linear problems of k obstacles* (abb. L_kM). Here, *reverse-planning* means that the robot plans from its goal to its start configuration, and from the last obstacle to move to the first. This contrasts heavily with all the NAMO planners presented until now, that conversely used a *forward-planning* strategy, that is: planning from the start to the goal robot configuration, and from the first obstacle to move to the last.



(A) Initial world state. The robot tries to reach the left corridor.

(B) The movement of the last obstacle to move (couch) is planned for first, with the subsequent navigation sequence to the goal.

(C) The area swept by the couch's movement and navigation to the goal (black area) is cleared by computing the table's movement.

FIGURE II.8: A slightly modified version of Fig.II.7c into an L_2 problem, showcasing Stilman & Kuffner's reverse-planning NAMO algorithm [46]. Read left to right for the planning process, right to left for execution.

[6] Also, Okada et al.'s algorithm is very likely to be unable to solve any problem that is not in the LP1 class, because of its graph structure akin to that of Stilman & Kuffner's. We say "likely", as, to the best of our knowledge, there is no sufficient documentation nor reference implementation available to ascertain this with absolute certainty.

[7] Assembly Planning can be considered a further constrained sub-problem of Rearrangement Planning, where one needs to consider the force of gravity and its impact on the assembled system's stability, and sometimes pre-specified order constraints. This problem often assumes that unassembled parts can be removed to "infinity", to reduce the amount of constraints to be taken into account and allow solutions to be found [45]. These problems typically also ignore the constraints of the robot/manipulator.

The last obstacle to be manipulated is selected using a heuristic A^*-based search similar to their previous algorithm [3], with a euclidean cost estimate, and affecting an extra one-time cost when traversing an obstacle[8]. While ignoring all other obstacles, a navigation path to a valid grasping configuration is found by means of an A^* search from the robot start configuration. Then, a manipulation sequence is computed by means of a Best-First Search that displaces this last obstacle, until a navigation path from the final robot grasping configuration to the goal is found by means of an A^* search (Cf. Fig.II.8a and II.8b). Logically, any obstacle that moved before this last one cannot be left in the area swept by the last obstacle in this process. Thus, the planner recursively plans manipulation and navigation motion in the same fashion as previously described, so that swept areas remain clear of obstacles, as illustrated by the table being cleared from the couch's swept area in Fig.II.8c. When the swept areas cannot be cleared at some point during the search, the recursive loop backtracks to select another obstacle manipulation order. This whole process (obstacle selection - recursive clearing of swept areas) is repeated until the goal is reached or all obstacle manipulation orders are considered.

While Stilman & Kuffner's new algorithm opened the door to solving harder problems in the new L_kM class, **it has not been proved L_kM-complete nor L_kM-optimal**. The main reason for this being, that while the algorithm does consider all obstacle orderings, *it does not consider all possible final configurations for each obstacle*. The authors however precise that **the algorithm could be L_kM-resolution-complete**, under the condition that it were modified so that it backtracks not only over object orderings, but also over choices of grasps and manipulation paths (which would be a significant additional computational cost).

II. 1.5 Sampling-based NAMO planning in continuous configuration space

In the previously mentioned papers, the computational infeasibility of complete planning for the full NAMO domain has lead to either not providing completeness guarantees [38, 2], or focusing efforts on sub-problem classes where algorithms based on graph search may guarantee resolution-completeness, within a reasonable amount of computation time [3, 46]. That is why in 2006 Nieuwenhuisen et al. [4, 48, 49], and in 2009 Van den Berg et al. [5, 50] explored *probabilistic completeness* - a weaker form of completeness that guarantees to find a solution if one exists, but under the condition that the allotted planning time is infinite.

The first algorithm proposed by Nieuwenhuisen et al. [4] consisted in building what they called an "action tree", where nodes represent either "grasp" or "manipulate" actions and their associated resulting world state (Cf. Fig.II.9). "Grasp" actions (round nodes in Fig.II.9) are equivalent to transit/navigation paths in the previous literature, and are computed using an A^* graph search over a Probabilistic Road-Map (abb. PRM). Correspondingly, "Manipulate" actions (triangle nodes in Fig.II.9) are equivalent to transfer/manipulation paths, and are computed using an RRT search (with a bounded number of node expansions). The children "manipulate" (triangle) nodes of a "grasp" (round) node each correspond to an individual node of

[8] Whereas the previously presented algorithm of Stilman's [3] added an extra cost for each traversed obstacle-colliding configuration (multiplying the cost by the number of robot configurations needed to pass through the obstacle). Also, this new heuristic allows collisions with more than one obstacle at once.

the RRT manipulation search of the grasped obstacle. One could say that the RRT manipulation search tree is "flattened" into action tree nodes (e.g. in the second action tree of Fig.II.9, the triangles directly under circle 1 are each a node of the RRT launched from the world state in circle 1). Only leaf nodes are expanded and only once, according to a priority determined by a probability of success. This probability heuristically evolves as the planning process progresses, and as the probability of a node is updated, its siblings' probabilities are adjusted too, and back-propagated to parents. This process firstly depends on whether the obstacle is *directly blocking* the shortest path to the goal computed on the PRM while ignoring obstacles. It also depends on whether the obstacle is *indirectly blocking* another obstacle's movement, which is known during the RRT planing of "manipulate" actions. The euclidean distance to the goal also affects the probability associated with a node.

Notably, and similarly to Chen & Hwang algorithm's [1], **re-grasps of a same obstacle are allowed**, as illustrated in the third action tree of Fig.II.9, where a second grasp (gray circle) node of obstacle 1 is a descendant of another (red circle). Re-grasp allows the robot to better maneuver around the obstacle as it manipulates it. This brings us to the authors' statement that the planner is capable of solving $\mathcal{LP}$ problems. Please note the difference in writing here, as it does not refer to Stilman's component-based definition given previously, but to the original rearrangement planning problem class definition of Ben Shahar & Rivlin [24] (that also inspired Stilman). Here, this class refers to "the set of problems that can be solved by a sequence of manipulations"[9]. The authors **conjecture that the planner is probabilistically complete (for the $\mathcal{LP}$ class) if robot motion constraints are holonomic** [49], but do not prove it, as quite a few following papers misleadingly conclude (e.g. [51, 52]).

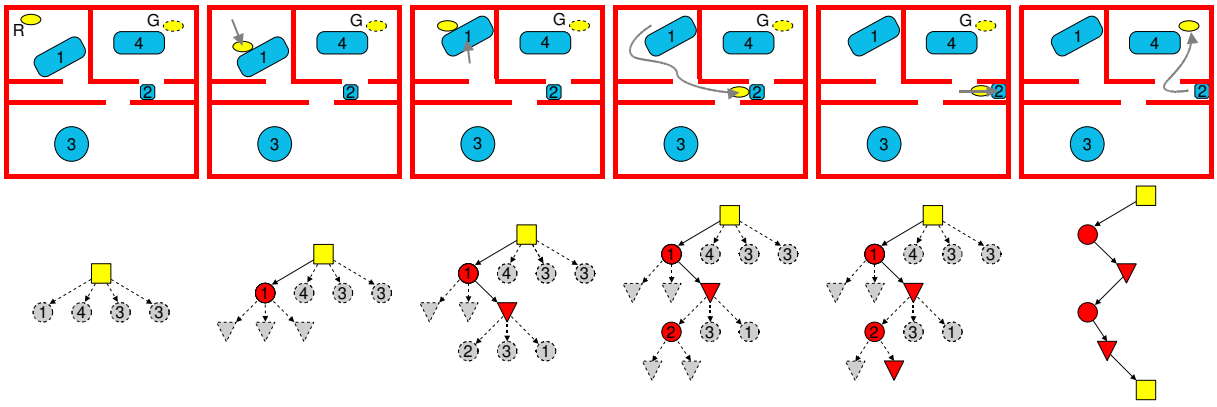


FIGURE II.9: Example L_1 NAMO problem solved by Nieuwenhuisen et al.'s algorithm [4]. From left to right, the vignettes depict the initial world state, then the alternating "grasp"/"manipulate" sequence for obstacles 1 then 2, before a direct navigation path to the goal can be found in the last vignette. Under each vignette is the "action tree", where round nodes are "grasp" nodes with the associated obstacle's number, and triangle nodes are "manipulate" nodes. The yellow squares represent the initial and goal states, and the red nodes are the selected nodes branch represented in the vignettes.

[9] To be understood as all problems that do not require simultaneous movement of more than one obstacle.

Actually, the subsequent paper of Van den Berg et al. [5] argues that the previously mentioned bounding of the search trees (necessary for proper backtracking over alternative obstacle grasps) is **a cause for (probabilistic) incompleteness for Nieuwenhuisen's algorithm** [4][10]. Consequently, they devised a new sampling-based algorithm and **provided a proof of its probabilistic completeness for the entire NAMO domain**. This proof actually only holds true if the NAMO domain is considered to be strictly restricted to Stilman's first definition, where *the robot can only manipulate one obstacle at a time, and assuming any obstacle can be grabbed from any configuration where the robot touches its envelope*[11]. However, *they could only prove its applicability for simplified axis-aligned environments where the robot and obstacles are limited to translations* (Cf. Fig.II.10). This limitation is due to their random tree-based algorithm requiring a custom memory-compact explicit representation of the robot's configuration space and free space components, that is to be computed and saved for every obstacle movement during planning. While we shall not delve into the details of this custom representation, its main point is that it allows to decouple the computation of the obstacles' movements from the explicit computation of the corresponding robot movements. That is, their custom representation allows the algorithm to first compute all the obstacles' movements regardless of the robot's, and only then infers the robot's movements from the planned obstacles' movements.

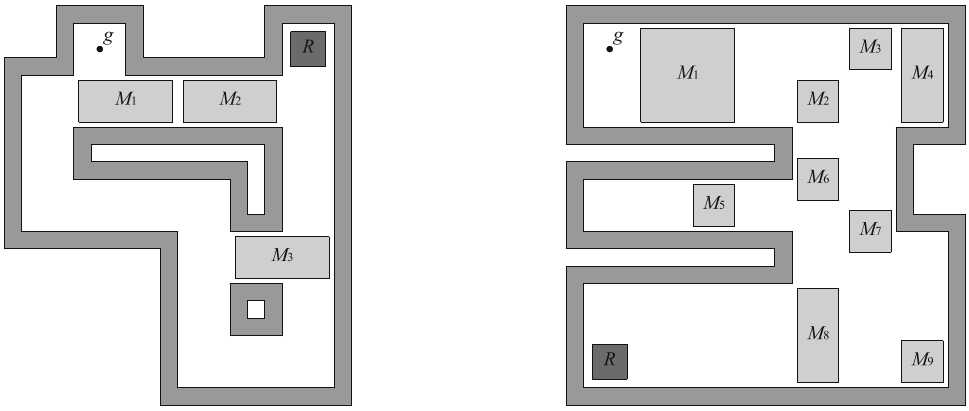


FIGURE II.10: NAMO problem examples solved by Van den Berg et al.'s algorithm [5]. (A) $NLNM$ problem where the robot must pass by M_2, then move it to the right, then move M_3 to the left, navigate through the bottom U-turn, move M_3 again but to the right, to finally move M_1 in the space left by moving M_2. (B) Axis-aligned version of an L_kM problem initially solved by Stilman & Kuffner's reverse-planning algorithm [46], where the smaller obstacles must be moved first to allow the necessary manipulation of M_1.

While this approach cleverly allows to address any of the previously described classes of problem mentioned before (such as the hard non-linear and non-monotone problem of Fig.II.10a), it still heavily relies on the two emphasized above hypotheses, and is definitely

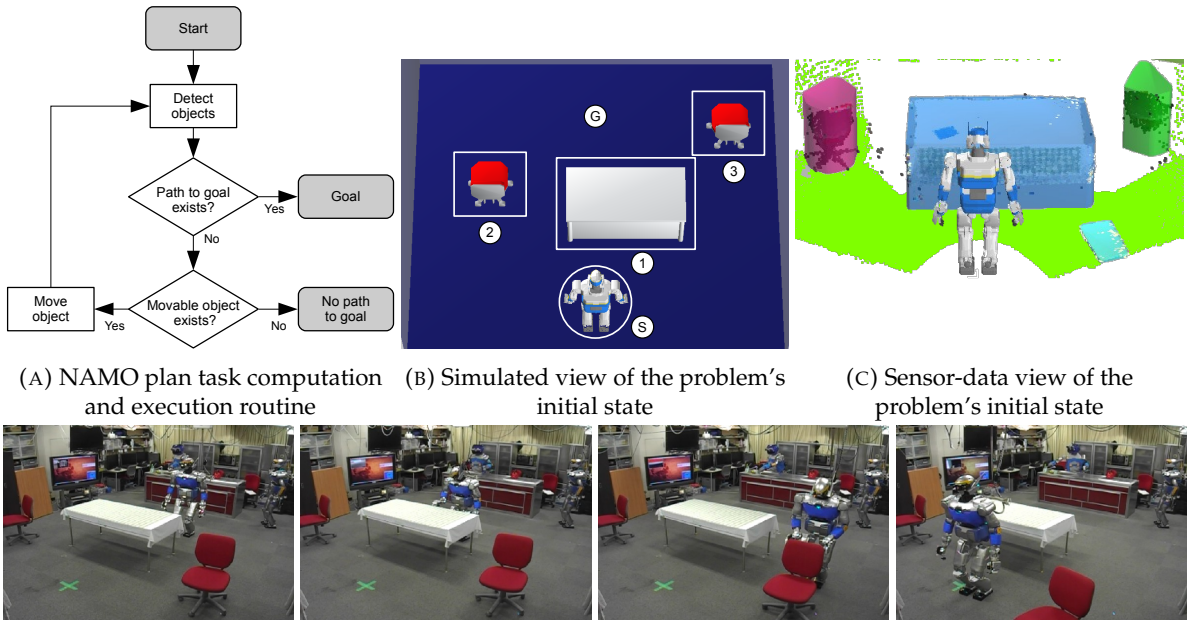
[10] Van den Berg was a colleague of Nieuwenhuisen, and although not identified as a co-author of [4], is mentioned in the Acknowledgements section of the paper.

[11] These hypotheses actually fit a lot with Nieuwenhuisen's NAMO interpretation of Ben Shahar's $\mathcal{LP}$ rearrangement planning class.

not trivially generalizable to more complex world geometries, which would come with a non-negligible time and memory cost (if at all mathematically computable). Finally, while this approach is probabilistically complete, its randomized nature produces very suboptimal plans, with respectively 214 and 23 899 obstacle movements (grasps) for Fig.II.10a and II.10b, while it took only 6 such movements for Stilman & Kuffner L_kM-planning algorithm [46] for the original scenario of Fig.II.10b.

II. 1.6 NAMO planning in unknown environments

All the previously described planners were designed under the assumption that **the robot has, at any time, an accurate knowledge of the entire environment** (obstacle position, geometry, type, mass, ...), and were thus mostly experimentally applicable in computer simulations. As mentioned earlier, the real world experiments of Okada et al. and Stilman & Kuffner [42, 44] relied on very precise and all-encompassing external sensors, human-made 3D models of the environment, in highly simplified experimental settings, to ensure that computed plans could actually be executed without failure. But for NAMO algorithms to be applicable in real-world settings, without prior knowledge of the environment, where the robot's main sources of information are its limited-view onboard sensors, these algorithms need to be able to use any partial information available to efficiently devise and update their plan.



(A) NAMO plan task computation and execution routine

(B) Simulated view of the problem's initial state

(C) Sensor-data view of the problem's initial state

(D) Initial real world state

(E) The robot fails to push the table (t=119s)

(F) The robot then pushes the chair (t=286s)

(G) The robot navigates to the goal (t=384s)

FIGURE II.11: Example NAMO problem resolution by Kakiuchi et al.'s algorithm [6]. In (C), the sensor's point cloud is segmented into prisms (obstacles) and floor points (bright green areas).

With this in mind, Kakiuchi et al. [6] published a first (reactive) approach under this sensing constraint in 2010. Their algorithm can be summarized as having the robot try to push the nearest detected obstacle in a single direction, until it either succeeds at some point in

computing a collision-free navigation path (with a bounded RRT) to the goal, or fails the manipulation / eventually bumps into another obstacle (Cf. Fig.II.11a). This reactive decision loop successfully solved the simple problem described in Fig.II.11b to Fig.II.11g, by segmenting real sensing data from the HRP-2 robot's color range sensor into primitive geometric shapes and their associated relevant grasping points (Cf. Fig.II.11c), and verifying movability using the robot's force-torque sensors as it manipulated obstacles. While they provide the first holistic approach to NAMO in unknown environments (from sensing, to planning, to execution in the real world), **their planning algorithm is arguably naive and computationally inefficient**. Indeed, the greediness of their nearest-obstacle choice heuristic, and the bounded RRT navigation search preclude any completeness or optimality guarantees, while limiting resolution capabilities to "one-manipulation" plans. Also, as a new RRT navigation search is called to verify goal reachability *at every obstacle manipulation step*, reasonable computation times are in fact only achieved thanks to the drastic robot action space restriction, at the cost of lower problem resolution capabilities.

While Kakiuchi et al. prioritized providing solutions to realistic sensing problems encountered while experimenting in a single real-world proof-of-concept setting, Wu&Levihn et al. devoted their efforts into designing a **more efficient, and locally-optimal planning algorithm** over several publications [33, 34, 35, 7] from 2011 to 2014, in multiple simulated environments. In contrast with Kakiuchi et al.'s algorithm that mindlessly pushes obstacles first to only then verify if a path to the goal appears, *this new algorithm carefully selects and computes the full motion for both the robot and obstacle to be manipulated ahead of execution*, similarly to the previous planners in the literature [3, 46]. This new algorithm relies on four essential hypotheses:

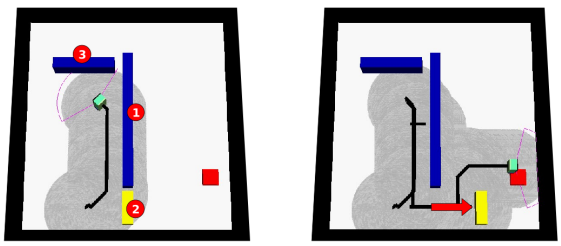
1. the robot can only compute plans with a single continuous obstacle manipulation[12];
2. unobserved space is to be considered free until discovered otherwise;
3. all obstacles are movable until manipulating them fails;
4. the robot is the single autonomous agent evolving in the environment.

The planner uses graph-search algorithms (A^*, D^*Lite, Breadth-First Search) over a grid discretization of the space, to compute the optimal navigation-only plan that avoids all obstacles, and plans each involving the manipulation of a single obstacle. The planner compares all these plans to guarantee that it will always choose the best-cost plan at any time during execution with its currently available environment information, under the hypotheses above - hence local optimality. As it tries to execute its computed plan, *the robot discovers new obstacles and updates its knowledge* about their shape or movability. When this new information invalidates the currently executed plan (future collision or failed manipulation because the obstacle was not movable), *it repeats the previously described planning process* to replace the invalidated plan. This process is illustrated in Fig.II.12a, where (Left) the robot executes a first plan trying to circumvent detected obstacles 1&2, but (Right) comes to face obstacle 3, and replans its course

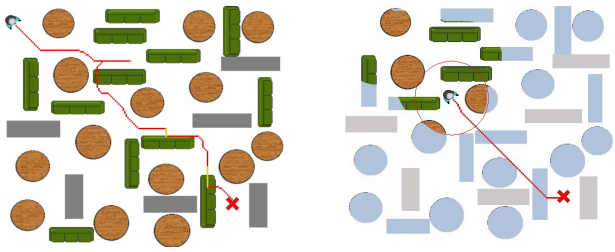
[12] As the robot discovers the environment and computes new plans, it may manipulate multiple obstacles during execution.

only to realize that avoiding it will be costlier than moving obstacle 1 - but eventually fails as it is a static obstacle, and finally replans to move obstacle 2 to reach its goal[13].

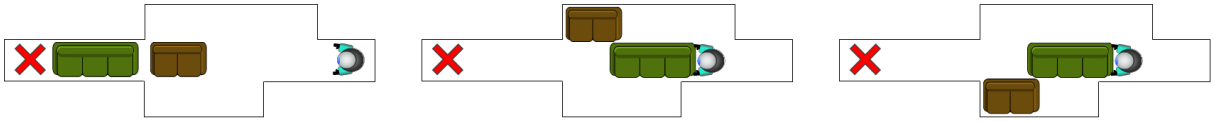
The first version of Wu&Levihn's algorithm [33] was actually not locally optimal, and it is the later publications of Levihn [34, 35, 7] that provided this guarantee, along with an extension to arbitrary-shaped obstacles, and a larger action space allowing obstacle manipulations in all axis-aligned directions instead of only single-direction pushes. These changes are reflected in Fig.II.12b, where the initial world state is overlaid with the path followed by the robot (i.e. the result of multiple replannings) in the left figure; the right figure shows the world perception after it pulled a first green couch to the right[14] (the next two couches are moved down). Importantly, one big contribution of this improved version is their efficient local opening detection algorithm [35] that can be used to reduce motion planner calls to verify the creation of a new path after manipulation (greatly improving computational efficiency)[15]. Still, the local optimality guarantee and computational efficiency of this algorithm come at the cost of problem resolution capability: while the algorithm is not a priori theoretically limited to a specific NAMO problem class, it can still be quite easily defeated with simple problems such as the one illustrated in Fig.II.12c. Depending on the algorithm's implementation (more precisely, the default order of manipulation directions), it may either choose to move the small brown sofa up (2nd subfigure) or down (3rd subfigure), respectively succeeding or failing the problem resolution, as obstacle ordering cannot be considered by the algorithm due to the one-obstacle per plan hypothesis.



(A) NAMO problem solved by the first version of Wu&Levihn's algorithm [33]. Robot in green, static obstacles in blue and black, movable obstacles in yellow, goal in red, explored space in gray, traversed path in black lines.



(B) NAMO problem solved by the final version of Wu&Levihn's algorithm [34, 35, 7]. Static obstacles in gray, movable obstacles in color, robot in gray and blue, goal as red cross, traversed path in red with yellow sections for manipulations.



(C) Hard L_2 problem for Wu&Levihn's algorithm. Please note that the brown sofa blocks the robot from seeing the green couch until the brown sofa is released.

FIGURE II.12: Example NAMO problem resolutions by Wu&Levihn's algorithm [7].

[13]The whole process is best understood with the associated video: https://youtu.be/oQZLbJHYr18

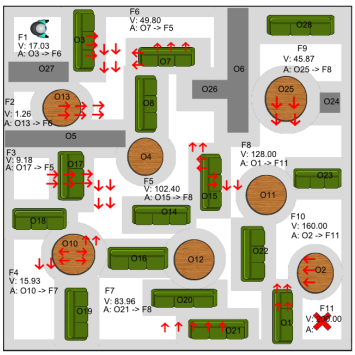
[14]The whole process is best understood with the associated video: https://youtu.be/3AvfPVzBb-s

[15]We provide an in-depth explanation and revised pseudocode formalization of Wu&Levihn's algorithm in the next chapter, Section III. 2.1.

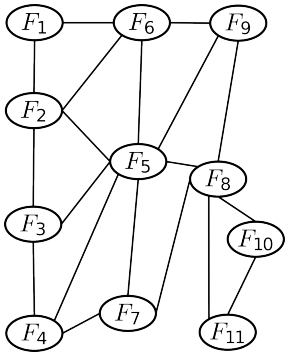
II. 1.7 NAMO planning under action and sensing uncertainty

Even while using only onboard sensors, the previously mentioned algorithms of Kakiuchi et al. and Wu&Levihn [6, 7] still **operated under a deterministic world and action model**. That is, like the rest of the NAMO planners before them, they assumed that sensor data was perfectly accurate, and that their planned actions would unfold exactly as expected. Only if an action were to actually fail to yield the expected world state during execution, would the world model be updated to forbid said action before re-planning would occur.

In 2013, Levihn, Scholz and Stilman proposed a framework to **deal with uncertainty in NAMO problems at the planning level** [8, 27], in order to bias decisions at plan time to choose policies (i.e. plans) that are likely to succeed. They cast the NAMO problem as a two-level hierarchy of Markov Decision Processes (abb. MDP), with a High Level MDP and a Low Level MDP. The High-Level MDP is built upon the adjacency graph of Stilman's first planner's paper [3] we previously presented, where nodes are free space components and edges are abstract actions that "create an opening to neighboring free-space" by manipulating an adjacent obstacle (Cf. Fig.II.13b). The Low-Level MDP's states are expressed in terms of robot & obstacles configurations, while actions are discrete axis-aligned translations (Cf. red arrows in Fig.II.13a).



(A) Initial world state with overlaid policies: High-Level in text, and Low-Level in red arrows indicating locally dominant directions



(B) Adjacency graph - Nodes are free space components, edges are neighboring obstacle(s) that may connect the components if moved.

FIGURE II.13: Example NAMO problem resolution by Levihn, Scholz and Stilman's decision theoretic algorithm [8]. In subfigure (A), the A value is the best action to execute (e.g. the best first action is to move O3 right, then down, to connect F1 to F6); The V value is the value-iteration algorithm's long-term expected reward for the state (i.e. the reward of passing through the connected component). Only low level policies corresponding to obstacles chosen by the high-level policy are shown for the sake of readability.

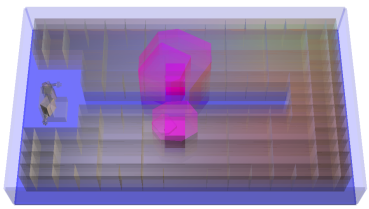
This structure only explicitly models action uncertainty, as perception uncertainty is incorporated through the use of object categories (e.g. "table", "table with locked wheel casters", ...), each associated with a probabilistic action model. That is, uncertainty is represented by both a probability distribution for an object to belong to a category, and by a probability distribution of 2D object displacements indexed by action and object category[16]. The task-level algorithm for solving the high-level MDP is a combination of the standard Dijkstra graph

[16]They chose not to model the problem with the usual approach of Partially-Observable MDP (abb. POMDP) to handle perception uncertainty, as they collapse this uncertainty in the Transition Model of the MDP for the sake of computational efficiency.

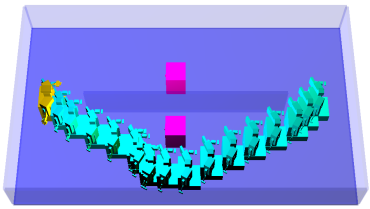
search algorithm to create the adjacency graph, with the stochastic MAX-Q Value Iteration Algorithm [53],while the motion-level algorithm for solving the low-level MDP is a Monte Carlo Tree Search (abb. MCTS) [54], that either terminates when an opening is achieved, or the pre-determined maximum search depth is reached. The algorithm alternates between computation of the high-level policy and low-level policies, using the reward values of the high-level search of the MAX-Q Value Iteration algorithm to dynamically bound the low-level MCTS's search depth, and the reward values of the MCTS to update the reward values of the high-level search. This two-way influence between the high and low-level policies searches allows the algorithm to focus computations on relevant high-level tasks (connecting components likely "useful" in reaching the goal), and their relevant portions, **achieving linear time complexity in the number of obstacles**. However, **this planner is not complete nor optimal** for any kind of NAMO problem class defined until now, and is **limited to the resolution of L_1 problem** because of its hierarchical decomposition of the problem.

Levihn, with the help of Leslie Pack Kaelbling and Tomás Lozano-Pérez, **ultimately addressed both action and perception uncertainty (including the absence of information) [9, 27], by applying their generic Belief Hierarchical Planner in the Now** (abb. BHPN) [55, 56], an algorithm meant to solve the overarching problem of combined Task & Motion Planning (abb. TMP or TAMP). Citing a recent survey on the matter [57], combined Task and Motion Planning is "the problem of planning for a robot that operates in environments containing a large number of objects, taking actions to move itself through the world as well as to change the state of the objects" - and is thus a superset of the NAMO problem. BHPN is an online algorithm, based on a backward planning task-level planning routine (similar to Stilman's L_kM planning algorithm [46]), and time-bound RRT calls for motion-level planning. It relies on a belief-based world model reflecting the history of observations and actions made by the robot, explicitly modeling perception uncertainty using ϵ-shadows [56] (akin to "fog of war" in video games) and probability contours [56] (volumetric overestimates of actual object geometries), as shown in Fig.II.14a. This model is used at plan time to bias plan decisions in order to minimize the risk of failure. This even includes explicitly planning for deliberate observation actions, so that the planner is guaranteed to have sufficiently reduced its uncertainty concerning the spaces where it is supposed to act. The planner relies on the main concepts of "Foresight" (leveraging of knowledge of future subgoals and beliefs about future observations to maintain correctness and improve optimality, illustrated in Fig.II.14b) and "Reconsideration" (deciding when to replan based on opportunity for plan improvement and computational cost of replanning, illustrated in Fig.II.14c). Levihn's work [9] mainly consisted in characterizing these two concepts for the NAMO problem, resulting in a planning-time efficient "selective replanning" strategy that allowed the algorithm to solve many NAMO problems of varying complexity, both in simulated and real-world environments[17]. **The planner does however not offer guarantees of completeness or optimality, but is theoretically not limited to any previously defined specific class of NAMO problem**.

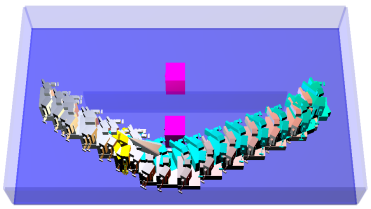
[17] As shown in multiple videos available at `https://lis.csail.mit.edu/bhpnNAMO/`



(A) Initial simulated problem state. Brown boxes are ϵ-shadows marking unobserved areas. Transparent purple boxes are probability contours representing observation uncertainty of obstacles.



(B) The robot uses "Foresight" to choose a plan passing through the bottom corridor, because of the lower uncertainty, even though the distance is slightly longer.



(C) As the robot executes the plan, the obstacle's probability contour is updated, and after "Reconsideration", a shorter plan closer to the obstacle is chosen (old plan in gray, new in green).

FIGURE II.14: Simulated NAMO problem solved by Levihn et al.'s BHPN-based algorithm [9].

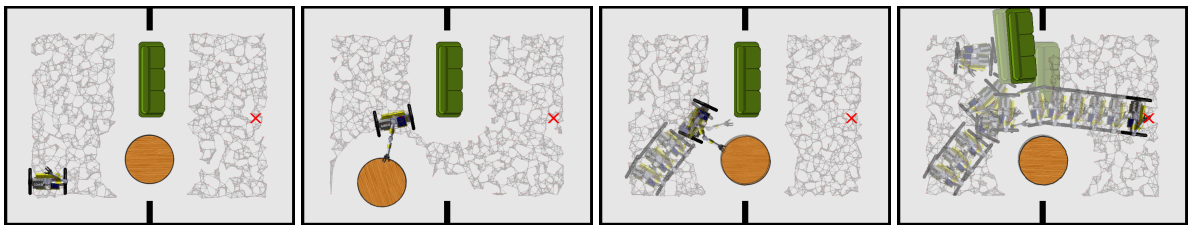
II. 1.8 Kinodynamic NAMO planning

Almost all the algorithms presented until now did not consider obstacle kinodynamics at plan time, and were solely built on geometrical world models. These models relied on the significant hypothesis that the robot's contact with the object would completely constrain its movement, so that, for instance, when the robot translated 10 cm forward, the obstacle would too - thus facilitating the study of the problem. All algorithms but one, actually: in his first L_1 planner [3], Stilman assumed a world model where obstacles were endowed with mass, moment of inertia and friction parameters. These parameters allowed obstacles to slip rotationally around the grasping point, as the robot would apply translational accelerations in any directions to its center of mass, allowing for complex movements. However, the difficulty for a real robot to sense and reason about the actual kinodynamic parameters of obstacles during execution easily explains why Stilman, and most of the following literature, fell back to a simpler world model.

Kinodynamic considerations were re-introduced in the NAMO problem by Levihn and Scholz in 2013 [10, 27], by extending their previously presented decision theoretic algorithm [8] to continuous configuration (state) and action (control) spaces. They kept the two-level hierarchical MDP problem decomposition, and the same task-level resolution algorithm, but to build the adjacency graph, they had to approximate the free space components in continuous space using Probabilistic Road-Maps (abb. PRM)[58] (illustrated as grey graphs in Fig.II.15), as the configuration space was no longer discrete[18]. Also, instead of using vanilla Monte Carlo Tree Search over discrete push actions, they used a Monte-Carlo simulation of Kino-Dynamic Rapidly Exploring Random Trees (abb. KDRRT) [59]. Instead of probability distributions over discrete push actions, they used probability distributions over obstacle kinodynamic parameters, such as mass, friction coefficient, position and orientation. That is, once an obstacle is selected for evaluation by the task-level routine, the Monte Carlo method is used to sample

[18]The state space is sampled randomly and biased to always include valid grasping poses around the movable objects: at the end of the PRM process, sampled grasping poses of a same obstacle that belong to different subgraphs of the PRM allow to detect that the obstacle is disconnecting free spaces components, and to create the relevant action edge in the adjacency graph (Cf. Fig.II.13b).

sets of kinodynamic parameters for this obstacle, and for each, a KDRRT search is run in a physics simulation, until an opening is created[19], or a maximum number of search nodes is reached. The planning-time physics-based simulation also allowed rewards to be more clearly expressed as functions of execution time and energy. The final novelty of this new algorithm is that the probability distributions over the obstacles' dynamics parameters (mass, friction coefficient, etc.) are updated as the plan is executed[20]. In the implementation they make of the algorithm, this "learning" step is however limited to updates of rotational parameters in case of rotation-only events, as illustrated in Fig.II.15. The authors provided theoretical and experimental evidence that despite the added complexity of this extension, the problem decomposition still keeps the algorithm's **computational cost linear with respect to the number of obstacles**, while retaining the same **resolution capability limitation to L_1 problems**.



(A) Initial configuration. The robot selects the lighter table for manipulation.

(B) Expected outcome: table has low probability of being constrained.

(C) Actual behavior: table rotates but does not translate. Robot increases probability of table being rotationally constrained to 98%.

(D) Based on the new information, the robot decides to move the couch.

FIGURE II.15: Example NAMO problem resolution by Levihn et al.'s kinodynamic & decision theoretic algorithm [10], showcasing a nonholonomic robot with 5 Degrees of Freedom (3DOF base and 2DOF arm).

Finally, in [11, 28], Scholz proposed a last iteration on the same decision theoretic algorithm [8, 10], that eventually allowed the algorithm to be **used on a real robot, in an environment with complex object dynamics**[21], as shown in Fig.II.16. To do so, this final version of the algorithm introduced two improvements. Firstly, a whole-body manipulation controller providing an abstraction of the robot's body into a low-dimensional action space (5 dimensions: 2 for the grasp point on the object, 2 for the desired object velocity, and 1 for the duration of the action). Secondly, they introduced a heuristic model-dependent manipulation policy, which constrained the set of achievable velocities and grasp points as a function of object dynamics. This heuristic model-dependent policy was based on anisotropic friction, a velocity constraint that allows separate friction coefficients in the x and y directions and an angular term. This is what allowed the authors to convincingly model the dynamics of the objects considered in their experiments: large objects such as chairs and tables, with or without wheel casters that

[19] Opening detection is done using a lower-dimensional RRT-connect search [60], that only considers the robot footprint, and is bound by a maximum number of nodes from the robot's configuration to a random configuration in the road map of the free space component to connect.

[20] The whole process is best understood with the associated videos:
`https://youtu.be/SFJbxUbuCO8` and `https://youtu.be/cQDpo9yxrCI`.

[21] The whole process is best understood with the associated videos:
`https://www.youtube.com/@toomanyquestion/videos`

may or not be blocked individually. While **the algorithm allowed adaptive behavior for previously unmanageable NAMO cases**, they recon that it was frequently faced with **a number of points of failure, mainly due to the randomness of many of the subroutines** that the PRM, the RRTs, the Markov Chain Monte Carlo (abb.MCMC)[61] used instead of the Monte Carlo KDRRT previously used in [10].

(A) Initial configuration. The goal is to reach the top right corner of the room.

(B) Expected solution: push the square table; but it is stuck - constraint learned.

(C) Expected solution: pull the long table; but it rotates unexpectedly because a wheel caster is locked - constraint learned.

(D) Expected solution: rotate long table: rotated successfully and opening found.

FIGURE II.16: Example NAMO problem resolution by Scholz et al.'s kinodynamic & decision theoretic algorithm [11], in a real-world setting. The robot has a differential drive base and two arms, totaling 17 Degrees of Freedom. Perception is provided by six overhead cameras tracking AR-tags[62].

II. 1.9 NAMO research since 2014

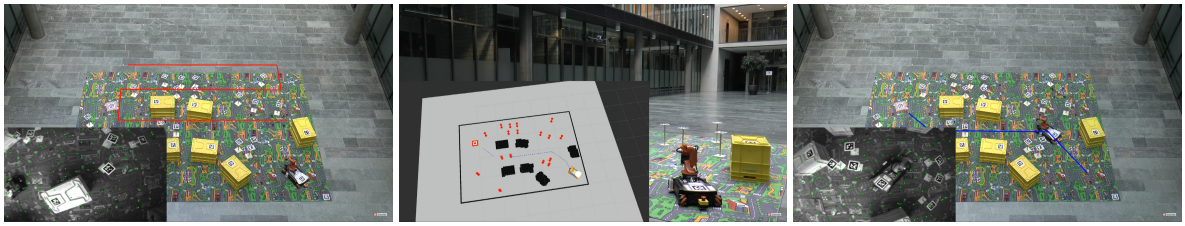
Mike Stilman disappeared in 2014, and with him, the GOLEMS research team he led at Georgia Tech[22], that spearheaded research about the NAMO problem, among other subjects. After the thesis defenses in 2015 of both his PhD students, Martin Levihn [27] and Jonathan Scholz [28], while the NAMO problem has continued to draw research interest, new algorithms rarely demonstrated new or superior theoretical properties, as we will present shortly. In a way, one could say that since then, research has shifted away from studying the NAMO problem itself, to using it more as a showcase context for other research.

A relevant first instance of this observation is the **2014** paper of Mueggler et al. [12], a demonstration of a guide quadcopter drone and ground manipulator robot, that needed to move small obstacles on a children's carpet to get to a goal configuration. The drone first scans the work area to find a ground target and map obstacle positions, then computes and sends the NAMO plan as high-level direction commands to the ground manipulator robot[23] (Cf. Fig.II.17). While they did link their experiment with the NAMO problem, they however admitted to dodging most of its inherent difficulty by only using one type of movable obstacle that could trivially be picked and placed without affecting the robot's 2D footprint. Thus, instead of using one of the previously described NAMO algorithms, they made an ad-hoc forward planning task-level algorithm derived from the A^* graph search algorithm, and used Dijkstra's algorithm as motion-level planning on a grid, with a custom path-smoothing procedure. Their algorithm is inherently limited to L_kM NAMO problems, as it only plans one

[22] The team website still exists at the following URL, and is a treasure trove of NAMO-related resources: `http://www.golems.org/oldindex.html`

[23] Videos available at `https://youtu.be/C5I190lzDdQ` and `https://youtu.be/OFPv3BegbFg`.

manipulation for each blocking obstacle at most. Also, although it seeks to minimize the execution time, the algorithm is not complete nor optimal for any of the previously defined NAMO problem classes, mainly because, contrary to Stilman's L_kM problem-solving algorithm [46], it does not consider interactions between obstacle manipulations.



(A) Drone mapping procedure uses AprilTags for obstacle identification and localization [63]

(B) Plan computed by drone viewed in Rviz before it flies back to the ground robot

(C) Drone follows the robot during execution, helps with localization

FIGURE II.17: Demonstration of Mueggler et al.'s heterogeneous drone + manipulator robot Multi-Robot NAMO system [12].

In 2016, Castaman et al. also proposed a new NAMO algorithm [64, 65] that did not build on any of the previous literature. Their algorithm is comparable in its approach to Chen & Hwang algorithm's [1], in that it used a global path planner that combined ideas from both the A^* the KPIECE [66] algorithms to produce an overall navigation plan, which was then locally modified by a custom manipulation planning routine in the places it intersected with movable obstacles. Incidentally, although it makes the algorithm's computational complexity independent of the number of obstacles, but also severely limits its resolution capabilities (as for Chen & Hwang's algorithm); it also makes it impossible to prove completeness or optimality for any of the previously mentioned NAMO problem classes.

The same year, Moghaddam & Masehian published [51] a NAMO algorithm that built a similar backward-planning task-level structure to Stilman's L_kM problem-solving algorithm [46], extending it to non-monotonous and non-linear problems (abb. NL, NM). They provide a theoretical proof for completeness, under the hypotheses that (1) *the robot only is allowed translation movements*, (2) *all obstacles are convex polygons and are priorly known*, (3) *the robot can grasp the obstacle at any point on their edges*, (4) *obstacles can only be manipulated by the robot*, (5) *there are no kinodynamic constraints on either the robot or obstacles*. This completeness only applies however to L_kM problems, and NL or NM problems that can be solved under the condition that "once an object is planned for, further planning for the same object is prohibited unless the robot displaces it to its pre-planned configuration". Beyond the above cited hypotheses, the fundamental algorithmic choice that allows the algorithm to maintain computational efficiency despite its larger problem resolution capabilities, is the fact that robot movement are computed on a visibility graph (abb. VG) that highly restrains the solution space. Since 2014, it is still the only new NAMO algorithm with novel theoretically demonstrated properties. However, these hypotheses and algorithmic choice make the algorithm hardly applicable or extensible to real-world NAMO problems, hence the authors only verified it in simulations.

In 2017 and 2018, Sun & Meng published a new sampling-based NAMO algorithm to showcase their real-time semantic mapping pipeline [67, 52]. Their algorithm drew some inspiration

from Wu & Levihn's [7] and Stilman's L_kM algorithm [46], but did not refer to the previous sampling-based NAMO literature cited above. Their backward-search task-level planner's structure, and their use of bounded RRT* [68] calls in it restrict the algorithm's solving capabilities to L_kM problems, and preclude probabilistic completeness for this problem class. But the whole point Sun & Meng's approach was to experiment in real-world scenarios, with real-world sensor data, assuming the robot only knows static obstacles beforehand. Their semantic mapping framework allowed them to detect object types (e.g. chair, table, etc.) and geometry, and choose the most appropriate manipulation strategy for each - including appropriate grasping points: pick&place[24] for likely lightweight ones, or planar holonomic push/pull on the floor for others. Finally, they proposed a new local opening detection algorithm as an alternative to using a motion planner call to verify the existence of a path, but do not evaluate its computational efficiency, nor compare it to Wu & Levihn's [35].

Finally, **in 2020** - at the time of our publications[25] [30, 69], the latest NAMO algorithm we could find was Wang et al.'s [70], based on the theory of affordances[26] and contact-implicit motion planning. All in all, their task-level planning routine is very similar to Kakiuchi et al.'s [6] (although they do not refer to it), with the same **limited resolution capability and lack of completeness or optimality guarantees**, but differs in two main ways. **The first difference** is that, while Kakiuchi's method only allows planar pushing/pulling, Wang et al. also allow for pick&place and planar pushing (same as Castaman et al. [64, 65] and Sun et al. [67, 52]). Both Castaman et al. and Sun et al.'s approaches relied solely on visual information for the manipulation strategy choice; Castaman et al. deriving the manipulation method to use from the object's point cloud size, Sun et al. deriving it from the object's semantic category deduced by image recognition. In contrast, Wang et al.'s algorithm used a modeling method [72] that derives affordances, first through visual detection, then by validating the expectation through physical interaction. **The second difference** with the naive approach of Kakiuchi et al., is that Wang et al.'s algorithm reasons about the final placement of the obstacle before starting the manipulation, as in the rest of the literature. The manipulation path is computed using an algorithm called Contact-Implicit Trajectory Optimization (abb.CITO) [73], simulating the interaction in the MuJoCo physics simulator [74], and using the Successive ConVeXification (SCVX) algorithm [75]. In contrast to this complex manipulation computation, navigation paths are simply computed using the A^* algorithm on a grid decomposition of the space (assuming the map of static objects is known beforehand). They validated their algorithm with two real-world experiments.

[24] Final placement of these small picked obstacles was planned not only for the floor plane, but every other reachable stable plane (tables and such) - to the best of our knowledge, their NAMO algorithm was the first to offer this capability.

[25] Whose contents shall be discussed in Chapter IV

[26] Affordances are possible interactions offered by the object to their user, as defined in the Theory of Affordance of J.J. Gibson [71]

II. 1.10 Conclusion

We synthesized the most relevant characteristics of the studied NAMO algorithms[27] in table II.1:

1. The first column reminds all relevant references associated with each algorithm.
2. The second column summarizes the various forms of prior knowledge that each algorithm requires as either **Full**, **Partial** or **None**. Most approaches require full prior knowledge of the environment, for some others a partial prior suffices (such as an occupancy grid of static obstacles), while only two algorithms rely solely on the robot's sensors: (Wu&Levihn, 2014) and (Kakiuchi, 2010).
3. The third column synthesizes the different ways the NAMO algorithm is informed of the obstacles' movability: it is either **Given** as prior knowledge, **Recog**nized through semantic segmentation of RGB(-D) sensors' data, or obtained through **Manip**ulation attempts. Unsurprisingly, most algorithms requiring full prior knowledge rely on being given that information, and most others rely on object recognition rather than only manipulation tentatives (likely because it is safer to move obstacles that appear movable than to move anything and everything).
4. The fourth column highlights that only the works of Levihn & Scholz - (Levihn, 2013-1 to 3) and (Scholz, 2016) - accounted for **State** and **Action** uncertainty at the planning level, while all other planners operate with deterministic state and action models.
5. The fifth column summarizes our previous comments on the problem resolution capabilities of each algorithm. Only (Wu&Levihn, 2014) provides local optimality guarantees, and only (Stilman, 2005), (Van den Berg, 2009) and (Moghaddam, 2016) provide completeness guarantees. All other algorithms surrender such guarantees in the name of computational efficiency.
6. The sixth column provides information about the cost(s) being optimized by each algorithm - something that has not been discussed much until now in the detailed descriptions of the algorithms. Among them, we find various estimates of the usual **D**istance, **T**ime and **E**nergy, but also more complex composite costs made from several of the previous ones for (Levihn, 2013-3) (Wang, 2020). Three algorithms - (Stilman, 2005), (Stilman, 2008) and (Kakiuchi, 2010) - first minimize the **N**umber of **M**oved **O**bstacles before considering the previously mentioned costs. In order to handle uncertainty, the planning algorithms of Levihn & Scholz's works - (Levihn, 2013-1 to 3) and (Scholz, 2016) also optimize the **P**robability of **S**uccess of their plans. The exception among all NAMO algorithms is (Van den Berg, 2009), that does not optimize any cost - resulting in very inefficient plans in terms of distance, time, energy or number of moved obstacles, as previously discussed.
7. The seventh column specifies whether a NAMO algorithm plans in **Disc**rete or **Cont**inuous configuration space - or both, in the cases of (Okada, 2004) and (Wang, 2020).

[27] Together with the algorithms developed in this thesis, for reference.

8. The eight column reminds the wide variety of existing algorithms used as subroutines of the NAMO algorithms: when abbreviated, the corresponding full names can be found in the Abbreviations glossary at the beginning of this thesis, or in the algorithms' descriptions above. Unsurprisingly, Random Tree-based algorithms are the most common, as they deal best with high-dimensional search spaces such as the ones found in NAMO problems.

9. The ninth column synthesizes the various assumptions made about the robot's object manipulation abilities. Most algorithms assume a fully-constraining **Grasp** of objects preventing any form of slipping, while some others constrain the robot's motion to solely **Push**ing the objects: (Kakiuchi, 2010), (Wu&Levihn, 2014), (Castaman, 2016), (Sun&Meng, 2018) and (Wang, 2020). Finally, some algorithms assume more complex models with specific Motion **Prim**itives computed through kinodynamic physics simulations, that may allow more freedom to the object's motion[28]: (Stilman, 2005), (Levihn, 2013-2), (Scholz, 2016). Most NAMO planners assume that the robot may only grasp an obstacle once during a plan, and only (Chen, 1991), (Nieuwenhuisen, 2008), (Van den Berg, 2009) and (Moghaddam, 2016) allow for regrasp. Finally, only (Chen, 1991) allows the robot to use an obstacle to move other obstacles, while all other NAMO planners assume this forbidden, likely for safety reasons.

10. Finally, the tenth column shows that most NAMO algorithms have only been experimented in simulations, but not in the real-world - which we shall comment more upon in the next paragraphs.

Beyond this summary, the first most relevant observation of this in-depth survey of NAMO algorithms is the **complete lack of reference implementations of all these algorithms, and severe lack of associated data for experimental environments**, their initial parameters or raw statistics. The current literature only provides human-readable illustrations and a select few heterogeneous aggregated statistics. The first consequence of this state of things is that studying and re-implementing any of these algorithms is a very arduous task. This likely mostly explains why, despite the existence of algorithms with relevant theoretical properties in the literature - such as (Stilman, 2005), (Stilman, 2008), (Van den Berg, 2009), (Wu&Levihn, 2014) -, no author ever experimentally compares their approach with previous work, although they sometimes draw some inspiration from these pre-existing algorithms. Another consequence of this difficult reproducibility, is that there is little to no verification of the claims made in the papers by other authors, leading, as we've discussed sooner, to sometimes misleading interpretations in the following literature. We try to address this reproducibility problematic in Chapter III.

The second relevant observation of this survey, is the **complete lack of social or multi-agent considerations** (be it other robots, humans, or anything else) before our study. These algorithms never consider the consequences of the robot's choices, beyond its interest to reach its goal while minimizing its displacement cost (be it execution distance, time, or energy). At best, only Stilman et al. [3] & Kakiuchi et al. [6] suggested in a single sentence each the idea of

[28] We discuss these notions in more details in Section III. 1.2.3.

taking object fragility into account, but have not applied it. This is why in the following Section II. 2 and Chapters IV and V, we explore the idea of a Social and Multi-Robot NAMO problem.

The third relevant observation, is that despite most algorithms presented here having the capability to move several obstacles in a same experiment, **almost all real-world experiments of NAMO algorithms to this day only ever required the movement of one obstacle to connect free space components**. Among those presented, the algorithms that study problems requiring complex obstacle manipulation ordering requirements to be solved - such as (Stilman, 2008) (Nieuwenhuisen, 2008), (Van den Berg, 2009) and (Moghaddam2016) - have only ever been experimented in simulation. This is true except for (Levihn, 2013-3)'s BHPN-based planner, where a single real-world experiment consisted in the manipulation of two obstacles to reach the next component. After almost two decades of study of the NAMO problem, this either means that robot sensing and manipulation capabilities are still insufficiently accurate to confidently solve bigger real-world NAMO problems, or maybe that real-world NAMO problems rarely exceed the complexity of the L_1 problem class. Maybe, the real challenge today is to better solve and integrate the solutions to all other upstream difficulties regarding world sensing and modeling, so that the algorithms may be used in real-world applications in human environments. While this thesis could also have addressed this third observation, we chose to focus our efforts on addressing the two other observations in the two previous paragraphs.

Finally, some other theoretical challenges are left open, such as creating:

- an actually resolution-complete algorithm for L_kM problems from (Stilman, 2008),
- a probabilistically complete algorithm for the full NAMO domain that is not restricted to the axis-aligned world model of (Van den Berg, 2009),
- or a better planner with actual theoretical guarantees that allows the manipulation of more than one obstacle at a time like in (Chen, 1991).

TABLE II.1: Synthesis table with main differentiating criteria

Reference	Prior	Movability	Uncertainty	Resolution capabilities	Cost	C-Space	Algorithms	Manipulations	Real-World
(Chen, 1991) [1, 38]	Full	Given	-	Limited to "plowing" paths	E	Disc.	Dijkstra	(Re)Grasp (with movement transfer)	No
(Okada, 2004) [2, 42]	Full	Given	-	Likely limited to L_1 problems	D or E	Disc. (Transit) and Cont. (Transfer)	RRT, Graph-Search Algorithms	Grasp	Yes
(Stilman, 2005) [39, 3, 43, 44, 19]	**Full**	**Given**	**-**	**L_1-Complete, can be made L_1-Optimal**	**NMO then E**	**Disc.**	**A*, Best-FS, Breadth-FS, DFS**	**Prim.**	**Yes**
(Stilman, 2008) [46, 47, 19]	Full	Given	-	Limited to L_kM problems, can be made L_kM-complete	NMO then D	Disc.	A*, Best-FS, Breadth-FS, DFS	Grasp	No
(Nieuwenhuisen, 2008) [4, 48, 49]	Full	Given	-	Limited to $\mathcal{LP}$ problems	D	Cont.	A*, PRM, RRT	(Re)Grasp	No
(Van den Berg, 2009) [5, 50]	Full	Given	-	$\mathcal{LP}$- Probabilistically Complete, limited to axis-aligned environments	None	Cont.	RRT	(Re)Grasp	No
(Kakiuchi, 2010) [6]	None	Manip.	-	Limited to "one-manipulation" plans	NMO then D	Cont.	RRT	Push	Yes
(Wu&Levihn, 2014) [33, 34, 35, 7]	**None**	**Manip.**	**-**	**Locally-Optimal for "one-manipulation" plans**	**D or T or E**	**Disc.**	**A*, D*Lite, Breadth-FS**	**Push, Grasp**	**No**
(Levihn, 2013-1) [8, 27]	Full	Given	Action	Limited to L_1 problems	PS	Disc.	Dijkstra, MAX-Q Value Iteration, MCTS	Grasp	No
(Levihn, 2013-2) [10, 27]	Full	Given	Action	Limited to L_1 problems	PS from T and E	Cont.	MCTS, PRM, KDRRT, RRT-Connect	Prim.	No

(Levihn, 2013-3) [9, 27]	Partial	Recog.	State, Action	No clear limit, no guarantees	Combined cost	Cont.	BHPN, RRT	Grasp	Yes
(Mueggler, 2014) [12]	Partial	Recog.	-	Limited to L_kM problems, MR	T	Disc.	A*, Dijkstra	Grasp (Pick&Place)	Yes
(Castaman, 2016) [64, 65]	Partial	Recog.	-	No clear limit, no guarantees	T	Disc.	KPIECE, A*	Grasp (Pick&Place) or Push	No
(Moghaddam, 2016) [51]	Full	Given	-	L_kM-complete under other hypotheses, but not limited to L_kM problems	E	Cont.	DFS, Dijkstra, VG	(Re)Grasp	No
(Scholz, 2016) [11, 28]	Partial	Recog.	Action	Limited to L_1 problems	PS from T and E	Cont.	MCMC, PRM, RRT	Prim.	Yes
(Sun & Meng, 2018) [67, 52]	Partial	Recog.	-	Limited to L_kM problems, Multi-Plane	D	Cont.	RRT*	Grasp (Pick&Place) or Push	Yes
(Wang, 2020) [70]	Partial	Recog. and Manip.	-	Limited to "one-manipulation" plans	Combined cost	Disc. (Transit) and Cont. (Transfer)	A*, CITO, SCVX	Grasp (Pick&Place) or Push	Yes
(Renault et al., 2019) [30] (Chapter IV)	**Partial**	**Manip.**	**-**	**Locally-Optimal for "one-manipulation" plans, with social taboos**	**D or T or E**	**Disc.**	**(Wu&Levihn, 2014)**	**Push, Grasp**	**No**
S-NAMO [69] (Chapter IV)	**Full**	**Given**	**-**	**L_1-Complete**	**NMO then E and S**	**Disc.**	**(Stilman, 2005)**	**Grasp**	**No**
(S)C-NAMO (Chapter V)	**Full**	**Given**	**-**	**MR**	**NMO then E (and S)**	**Disc.**	**(Stilman, 2005)**	**Grasp**	**No**

Legend: Manip. = Found through manipulation; Recog. = Found through visual recognition; '-' = No uncertainty management in planning; D = Distance; E = Energy; T = Time; NMO = Number of Moved Obstacles; PS = Probability of Success; Disc. = Discrete; Cont. = Continuous; Prim. = Motion Primitives; MR = Multi-Robot

II. 2 Social & Multi-Robot considerations in NAMO-related problems

In the previous Section II. 1, our thorough review of NAMO research has revealed a lack of concern for social or multi-robot considerations in the existing literature. However, Social Robotics and Multi-Robot coordination are far larger and more abstract problem domains, with thousands of papers addressing them in a huge variety of ways. Thus, the goal here is certainly not to provide as exhaustive a review of existing work as we did for NAMO. As we presented our review of NAMO algorithms, we also mentioned other robotics problems closely related to NAMO: Navigation Planning, Manipulation Planning, Rearrangement Planning, Assembly Planning, and the overarching domain of combined Task and Motion Planning. Thus, we will instead focus on providing examples of how social and multi-robot constraints have been accounted for in these neighboring problems, as a way to introduce relevant concepts for the next chapters and to better situate our work.

II. 2.1 Socially-aware robot navigation, task and motion planning

The official webpage of The International Journal of Social Robotics defines **Social Robotics** as "the study of robots that are able to **interact** and **communicate** among **themselves**, with **humans**, and with the **environment**, within the **social and cultural structure** attached to its role" [31]. This same page provides a long list of topics of interest to the field of Social Robotics, starting from "Affective and cognitive sciences for socially interactive robots", and ending with "Socially-aware robot navigation, task and motion planning".

Given the list of NAMO-related problems above, it is naturally the particular topic of **Socially-aware robot navigation, task and motion planning** that is most closely related to our own topic of interest of NAMO. Let us first start by discussing **Socially-aware robot navigation**, specifically.

II. 2.1.1 Socially-aware robot navigation

Socially-aware robot navigation, is synonymously called Human-aware navigation or Social Navigation in the multiple surveys addressing it [14, 13, 76, 77, 78, 79, 80]. To the best of our knowledge, none of the papers cited in these surveys ever considered having the robot operate in modifiable environments with movable obstacles, and planning obstacle manipulations to create a path or improve the cost of an existing path. Although we could not find any traces of the NAMO problem in existing Social Navigation literature, said literature remains very relevant to our subject of study. Hence, in this section, we shall cherry-pick useful notions and definitions from the first (and most cited) two surveys [14, 13], that will eventually serve us in Chapter IV.

Social navigation has been loosely defined in a first survey by Kruse et al. as "the intersection between research on human-robot interaction (HRI) and robot motion planning" [14].

From this definition, the survey identified the three following three main challenges of social navigation:

- **comfort**: "the absence of annoyance and stress for humans in interaction with robots";
- **naturalness**: "the similarity between robots and humans in low level behavior patterns";
- **sociability**: "the adherence to explicit high-level cultural conventions".

Of course, the fundamental **safety** requirement of "classic" navigation (guarantee of no collision) remains. The three main challenges listed above can actually be understood as *perceived safety*: beyond guaranteeing the impossibility of harm to humans, social navigation introduces the idea that the robot must not *feel threatening*, hence the three aforementioned challenges.

In a second survey [13], Rios-Martinez et al. made the observation that "concerning the field of robot navigation, proxemics is the most investigated tool to improve the robot's sociality". The term of **Proxemics** was first coined in 1963 by Edward T. Hall, a cultural anthropologist, defining it as "the interrelated observations and theories of humans **use of space** as a specialized elaboration of culture" [81]. In their survey, Rios-Martinez et al. defined the term as "the study of **spatial distances** individuals maintain in various social and interpersonal situations - these distances vary depending on environmental or cultural factors". This resulted in a new definition of social navigation focused on the notion of space, that shall serve as a basis for this thesis : "A socially-aware navigation is the strategy exhibited by a social robot which identifies and follows **social conventions** (in terms of management of space) in order to preserve a **comfortable** interaction with humans. The resulting behavior is **predictable**, **adaptable** and **easily understood** by humans". This definition is reflected in their abstract illustration of the structure of a socially-aware navigation system (Fig.II.18).

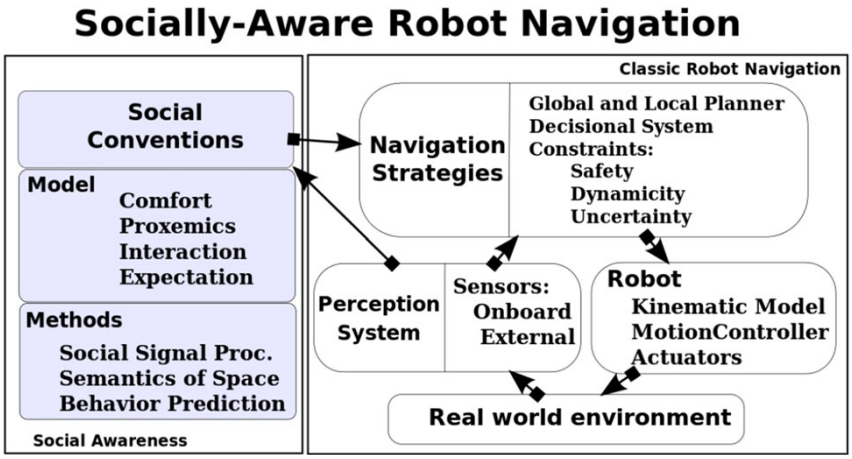


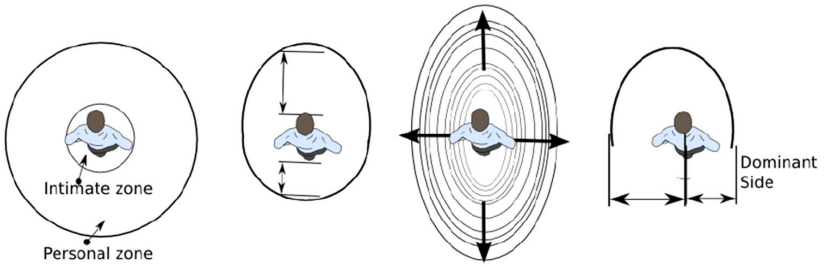
FIGURE II.18: Figure from Rios-Martinez et al.'s survey [13], depicting "The most important components of a socially-aware navigation system", a composite system with a social layer applied upon "classic" navigation elements.

Rios-Martinez et al. have classified social spaces relevant to robot navigation into four main categories: (1) the ones related to a single person, (2) to groups of people interacting, (3) to human-object interaction and (4) to human-robot interaction. Let us first present these four

categories, so that we may discuss whether they raise questions specific to a Social Navigation Among Movable Obstacles context.

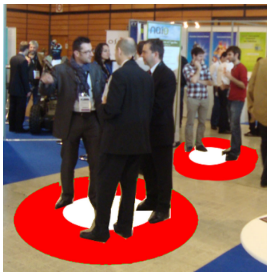
According to Rios Martinez et al., within the first category (1), **personal spaces**, is "the most popular proxemics model used in robotics", which they define as "the region around humans that they actively maintain into which others cannot intrude without causing discomfort" [13]. As illustrated in Fig.II.19a and II.19b, there are many formulations of these spaces, varying in shape, size, that may eventually change depending on a variety of parameters such as context, speed (either of the human or of other surrounding agents), nature of other agents, etc. These spaces are human-centered, and are generally mainly derived from human position and orientation and social cues such as gaze direction and body posture.

(A) Real-world example of people respecting each other's personal space (blue discs)



(B) A non-exhaustive illustration of different personal spaces definitions in the literature. From left to right, concentric circles, egg shape (bigger in the front), ellipse shape, smaller shape on the dominant side.

(C) Real-world situation of the Information Process Space



(D) Real-world example of social spaces created by a group: the interaction area (O-space, white disc), and area of human placement(P-space, red ring)

(E) Real-world example of an activity space. A human taking a picture creates a zone that should not be penetrated: the field of view between the camera and the object.

(F) Real-world example of an affordance space. The bus schedule offers a reading affordance to humans, creating a reading affordance space in front of it.

FIGURE II.19: Illustrations from Rios-Martinez et al.'s survey [13], depicting a variety of social space types and their variations.

The first social space category (1) also includes the notion of Information Process Spaces (abb. IPS), defined as "the space within which all objects are considered as potential obstacles when a pedestrian is planning future trajectories", illustrated in Fig.II.19c. Unless a robot wishes to engage interaction with a walking human, like for Personal space, it should avoid penetrating this zone so as not to affect the human's trajectory.

The second category of Social spaces for groups of interacting people (2) requires to first distinguish two types of interaction: focused and unfocused. Rios-Martinez et al. define **Focused interaction** as the situation "when individuals agree to sustain a single focus of cognitive and visual attention", while **Unfocused interaction** can be understood as the opposite situation of "an individual [simply] being in another's presence". A human, or a robot avoiding the previously presented Personal Space or Information Process Space of other humans while navigating is a perfect example of unfocused interaction, for instance. Focused interaction between humans, such as conversations, creates new social spaces, that grow beyond the addition of individual Personal Space, as illustrated in Fig.II.19d. Again, unless a robot wishes to engage in the focused interaction, it should avoid penetrating these zones so as not to bother implicated humans.

The third category of Social spaces for human-object interaction (3) is divided by Rios-Martinez et al. into two - in fact tightly related - categories: Activity and Affordance Spaces. An **Activity Space** is defined as a space "linked to actions performed by agents" (illustrated in Fig.II.19e), and is a term proposed by the roboticists Lindner & Eschenbach [82], derived from the term of "activity footprint" proposed by geographers Ostermann & Timpf [83]. Similarly, they defined an **Affordance Space** as a space "related to a potential activity provided by the environment" (illustrated in Fig.II.19f), following the previously mentioned Theory of Affordances of J.J. Gibson [71]. In their words, an Affordance Space can be considered a *potential* Activity Space. Like Personal Spaces, Affordance and Activity Space can be defined with a variety of shapes, sizes depending mainly on the action(s) they are related to, but also the objects they are derived from, their type, geometry, possible states (e.g. on/off for a light switch). While navigating through an Activity Space is to be avoided unless the robot needs to trigger a focused interaction with the human(s), an affordance space can generally be crossed without causing any disturbance - blocking it indefinitely would however not be socially accepted.

Finally, the last and fourth category of Social Spaces for human-robot interaction (4) can be synthesized with the observation that "it seems that the behavior of people sharing spaces with robots is not so different from the way they behave with other people" [13]. As humans expect social robots to behave like humans, one could consider that the social spaces presented until now also apply for robots.

Social Spaces are often represented using cost functions, that associate a value to all combinations of robot actions and action contexts, as shown in Kruse et al.'s survey [14]. Rather than strictly forbidding robot navigation in any part of the Social Spaces, using a cost function allows social navigation algorithms to ponder the movement cost of the robot against a cost of entering the Social Spaces. As a general rule, the more the robot's path penetrates into any social space, the higher the cost: this can be seen in the various cost function examples given in

Fig.II.20. These cost functions are oftentimes discretized into cost maps (Cf. Fig.II.21), generally square grids, associating each cell with a temporary occupancy cost. This facilitates the combination of the wide variety of cost functions for the different Social Spaces that may overlap one another, and allows for faster computations and simple visualizations, but as a drawback, may significantly oversimplify the problem, while raising a new problem of choosing proper weights to ponder costs with one another.

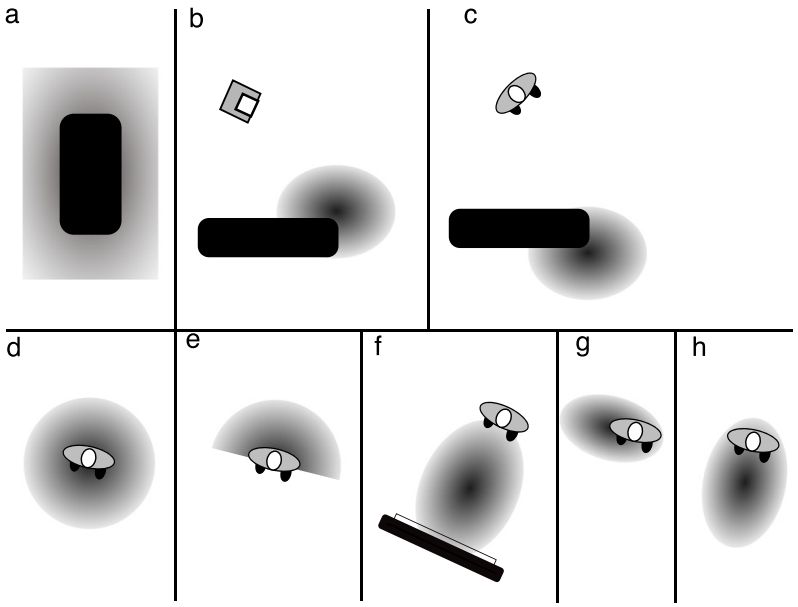
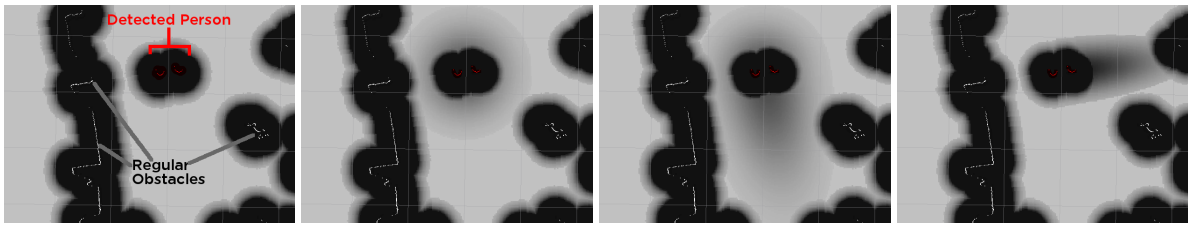


FIGURE II.20: Figure from Kruse et al.'s survey [14], roughly visualizing costmaps for different social spaces, represented as gray gradients (the darker, the higher the cost and need to avoid the space). Obstacles are in black, robot is a square (in b), human is an oval shape. Costmaps respectively represent, (a) object padding, (b) object occlusion, (c) hidden zones, (d) basic comfort distance, (e) visibility, (f) activity space, (g) pass on their left, (h) space ahead for moving (the human is moving towards the bottom-left corner). Shapes, size and cost distribution vary depending on the cost function, and can be straightforwardly combined.



(A) Obstacle costmap with object padding (Cf. Fig.II.20.a)

(B) Basic comfort distance increases cell costs around stationary human (Cf. Fig.II.20.d)

(C) Space ahead moving human increases cell costs in the movement's direction (Cf. Fig.II.20.h)

(D) Cell costs are increased on the human's right side to encourage robot to pass on the left (Cf. Fig.II.20.g)

FIGURE II.21: Actually implemented discretized social costmap models in the standard ROS Navigation Stack [15]. Layers can be combined through addition (bound with a maximum value).

Now, how would these Social Spaces designed for "classic" navigation, apply to a hypothetical Social Navigation Among Movable Obstacles ? NAMO is still first and foremost a navigation problem, which main focus is going from a start to a goal configuration. As such, like for "classic" navigation, a social NAMO planner should seek to avoid computing robot motion paths that cross the various social spaces mentioned above, as to not disturb humans, unless some focused interaction with a human being is required.

As explained in the previous section, where NAMO actually differs from "classic" navigation is in the fact that it may displace obstacles in the environment: obstacles that will then occupy a new space indefinitely. If a robot *momentarily* passing through any of the previously mentioned social spaces is already considered to be a disturbance in an unfocused interaction context, an obstacle being *indefinitely* placed in the same social spaces is bound to be considered an even greater disturbance. Imagine talking to someone or watching the television, and a robot suddenly left some cardboard box or chair in between: surely, you would feel far more bothered than if it quickly passed and left. Similarly, in the same way robots are expected to go to specific zones in social spaces when engaging in focused interaction, one could expect that moved objects may need to be placed in specific spaces.

Personal Spaces, Information Process Spaces and Activity Spaces are all human-centered social spaces. That is, not only do they require the actual presence of human beings in the environment to exist, they are inherently spatially tied to the humans' position in space. As such, when humans move, these spaces will move along with them, and may eventually disappear when their associated activity is over. Affordance spaces do however not disappear, as long as the objects that generate them remain in the environment. Any time these spaces would be used by a human (thus temporarily becoming activity spaces), the disturbance caused by an obstacle left in an inappropriate affordance space by the robot would add on, over and over again - until it became so uncomfortable the human would bother to move the obstacle elsewhere.

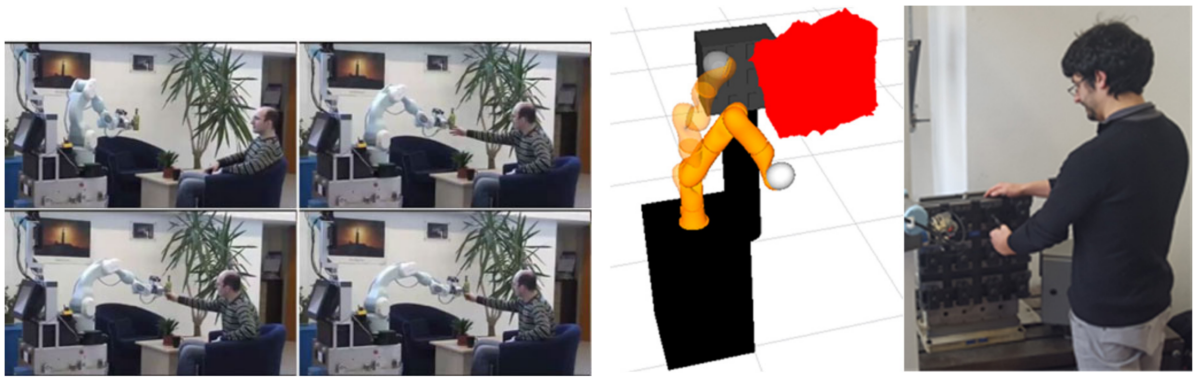
According to all the surveys on social navigation we have cited here, experimenting and evaluating human-centered models is hard. For real-world experiments, finding human participants and setting up the lab and robot for meaningful experiments is extremely time-consuming and requires minute consideration of biases such as participants fatigue or adaptation to the experiment. For simulated experiments, the difficulty lies in properly simulating relevant human behaviors, which is already not a trivial endeavor when assuming "classic" navigation tasks only, even with existing tools such as PedSim [84]: simulating a convincing social NAMO experiment with humans would require to simulate their capability to interact with obstacles too.

These two observations, with the conclusion in the previous section that the core difference between NAMO and adjacent problems lies in the choice of obstacles to move and their placement choice, motivate our study of a Social NAMO through the lens of affordance spaces in Chapter IV. We believe that the first - and most relevant - consideration towards a socially-aware NAMO is to ensure that even in the absence of humans at the time of NAMO plan execution, the robot should carefully select obstacle placements so as not to disturb humans when they will inevitably enter the space.

II. 2.1.2 Socially-aware robot task and motion planning

In contrast with the abundant Social Navigation literature, we could only find a few papers describing Social/Human-aware variations of the other problems related to NAMO (Manipulation Planning, Rearrangement Planning, Assembly Planning, and combined Task and Motion Planning). All of these papers only addressed situations of direct/focused human-robot interaction, making it all the more relevant to explore the unfocused interaction that is caused by the disturbance created by obstacle manipulations in NAMO.

More precisely, a lot of the socially-aware Manipulation Planning literature focuses on the specific task of computing and proper hand-over motion sequences [85, 86, 87, 88, 89, 90, 91, 92, 93] - that is, the task where the robot must give a human a small object currently in its grasp (Cf. Fig.II.22a). The presented situations and solutions do not quite fit a NAMO setting, as NAMO is mainly concerned with obstacles that block the robot's way, and generally of sizes comparable to that of the robot, requiring the robot to carefully compute not only it's body's path, but the obstacle's too. Still, the model of the human interaction space may be useful to address the concern of collaborative manipulation of obstacles in a NAMO problem. The rest of the research focuses on collaborative manipulation in close proximity / a shared workspace (e.g. moving individual objects on a table without colliding/interfering with each other's action, Cf. Fig.II.22b), mainly through prediction of the human's actions [94, 95]. This last focus is shared with the socially-aware Assembly Planning literature [96, 97, 98], and the socially-aware Combined Task/Motion Planning literature [99, 100]. Again the presented situations and algorithms do not fit a NAMO setting either as the workspace is fairly small, in contrast with the much larger environments considered in the NAMO literature. None of these works address the choice of obstacles to move and subsequent placement choice necessary in a NAMO problem.



(A) Example handover task from [90].

(B) Example collaborative assembly planning task from [96].

FIGURE II.22: Illustration of the two main situations addressed in the socially-aware variants of NAMO-related problems other than Social Navigation

II. 2.2 Multi-Robot coordination

In the same way that we needed to define, in the previous Section II. 2.1, specific vocabulary and terms related to Social Robotics and Socially-aware robot navigation, task and motion planning, we will need to summarize relevant vocabulary for Multi-Robot coordination. Hence, Section II. 2.2.1 will introduce these necessary terms, which will help us discuss existing Multi-Robot works that somewhat resemble NAMO, and conclude as to how we will formulate the Multi-Robot NAMO problem studied in this thesis in Section II. 2.2.2. The reader already accustomed to the vocabulary may thus want to skip through the first section.

II. 2.2.1 Definitions

Let us start by clarifying an essential term that will be used throughout this section and thesis: we call a **Multi-Robot System** (abb. MRS), a group of several robots acting in a common environment, also called workspace. In their recent 2021 survey [16], Verma & Ranga (which we will refer to as "they" in this section) propose an up-to-date classification of such systems, derived from previous surveys in the literature and most recently published works, in order to situate existing MRS on a continuum between opposite terms. These terms are summarized in Fig.II.23, and clarified below (unless stated otherwise, quoted text comes from [16]). Before continuing further, we shall emphasize on the notion of **continuum**: a Multi-Robot System rarely only fits a single one of the opposing terms we are about to present. More often than not, according to the survey, actual systems in the literature are in an in-between, that may fit with one term in some aspects, while some other aspects may be qualified as the opposite. Still, many systems fit more with one term than with its opposite, and may be categorized as such - while still bearing in mind that all is not black or white.

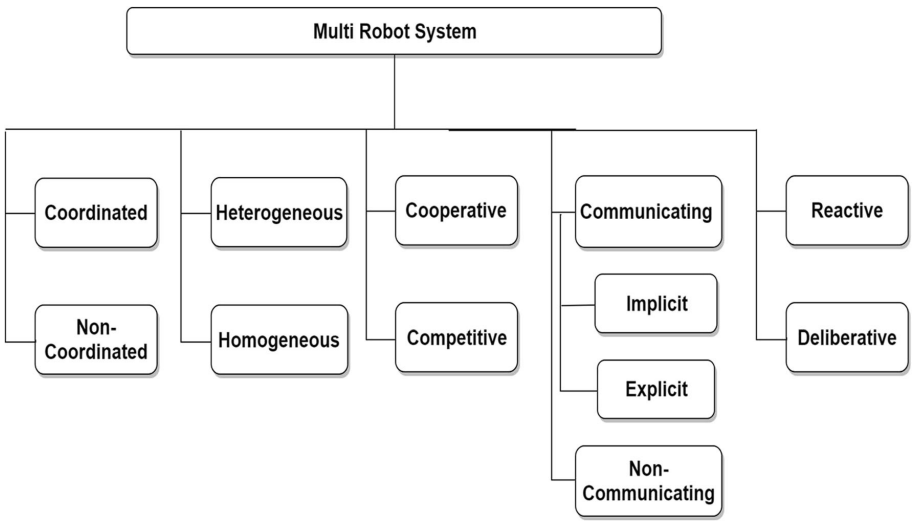


FIGURE II.23: Multi-Robot Systems (abb. MRS) Taxonomy proposed by Verma & Ranga in 2021 [16]

Coordinated/Non-Coordinated MRS: Multi-Robot Systems are generally **Coordinated** in that they aware of each other - through sensing, active communication, or by operating with a shared set of pre-determined rules [101]. This property is essential if the robots share common resources, such as space for navigation, in order to prevent **conflicts**: concurrent access to a same resource (e.g. in navigation: a collision, in NAMO: trying to manipulate a same obstacle). Conversely, **Non-Coordinated** MRS are not aware of each other.

Homogeneous/Heterogeneous MRS: Verma & Ranga state that most MRS in the literature are **Homogeneous**: with the same hardware and software. However, they underline that interest in studying **Heterogeneous** MRS, with varying degrees of dissimilarity in either software, hardware, or both, is growing, as robotics grow more popular and an ever-increasing variety or robotics systems are released. Heterogeneity brings more versatility (increasing the diversity of executable tasks), but at the cost of higher coordination complexity and difficulty compared to homogeneous systems.

Cooperative/Competitive MRS: MRS exhibit collective behavior similar to that of human society, that can either be Competitive or Cooperative [102]. Verma & Ranga define **Cooperative** behavior as "interaction among robots to execute a task along with increasing the system's overall utility", and inversely, **Competitive** behavior as "the case in which multiple robots compete among themselves in order to satisfy their own interest".

Communicating (Explicit/Implicit)/Non-Communicating MRS: A Cooperative behavior implies **Communication**, that is, any way robots can exchange or sense some information about each other. In a **Non-Communicating** MRS, "there is no information available about another robot to any other one", making coordination only possible through a shared set of pre-determined rules (e.g. follow a line on the ground, always turn right, etc.). In **Communicating** MRS, the way information can be shared is either classified as **Explicit** (syn. Direct), that is, using "additional communication hardware, a dedicated device for signals that can be understood by other team members", or **Implicit** (syn. Indirect), that is when "robots obtain information about other member robots through the environment". This notion is comparable to how Social Robotics differentiate between Focused and Unfocused interaction, as discussed in Section II. 2.1.1.

Reactive/Deliberative MRS: Finally, MRS can be classified according to the way they deal with changes in the environment as they execute their tasks [103], similarly to the domain of navigation in unknown environments mentioned in Section II. 1.6. An MRS is called **Deliberative** "if robots can cope up with any change in the environment by some approach to restructure the overall team behaviors" - that is, reconsidering the global plan for all robots through explicit communication. Conversely, an MRS is called **Reactive** if "every single robot copes with the changes in the environment by giving a robust solution to re-organize its own task with the purpose of completing its initially given goal" - that is, locally adapting the individual robot's plan to deal with the change.

Preliminary conclusion: Unless robots were explicitly bound to strictly separate spaces, a Multi-Robot NAMO problem inherently needs to be coordinated to some degree as to preserve the fundamental NAMO guarantee of a lack of collisions. According to Verma & Ranga's survey, *Coordination* has been defined in multiple ways in the literature, sometimes as a synonym of *Cooperation*, and sometimes not. We shall retain the following definitions in this thesis: **Coordination** is "the mechanism used for cooperation"[16], and **Cooperation** is the action by which agents take into account the other agents' actions so that "there is an increase in the total utility of the system" [104]. Coordination approaches can be categorized based on various parameters, of which we shall introduce below the most relevant ones according to Verma & Ranga, that are summarized in Fig.II.24.

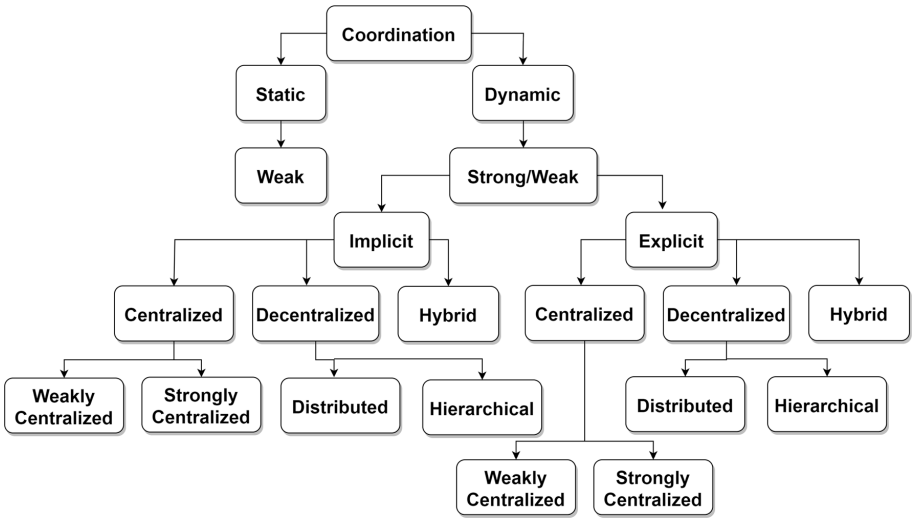


FIGURE II.24: Multi-Robot Coordination Taxonomy proposed by Verma & Ranga in 2021 [16]

Static/Dynamic Coordination: As mentioned previously, robots can be coordinated simply using pre-determined rules or conventions that will ensure no conflicts may occur during navigation (e.g. follow a line on the ground, respect automated signal lights, etc.): this type of coordination is referred to as **static** [105]. All other coordination strategies are classified as **dynamic**, as they are adapted on the fly during execution, depending on the system's state and the perception the robot has of it.

Weak/Strong Coordination: Coordination can also be broadly described as **Strong** (syn. Tight), "a method of coordination that is based on a coordination protocol" [103]. The more information and discussion/negotiation are needed to achieve the coordination strategy, the stronger the coordination. That is why Static coordination is conversely always qualified as **Weak** (syn. Loose), while Dynamic coordination strategies that require the robots to share more than their current state or goal, that is, up to their entire plan and/or decision process, are qualified as strong.

Implicit/Explicit Coordination: Coordination can also be classified in the same way as MRS on the basis of their means of communication. **Implicit** coordination uses perceptions of environmental changes, including observations of other robots (i.e. implicit communication), to create emergent behaviors that allow to avoid conflicts. **Explicit** coordination uses intentional communication (i.e. explicit communication) to achieve cooperation through a more or less democratic consensus. Implicit coordination is more robust to communication systems failure or unavailability, and scales better with growing number of robots in terms of computational requirements, but cannot guarantee optimality alone, and may fail in solving tightly constrained problems (e.g. many robots in comparatively little space) [16].

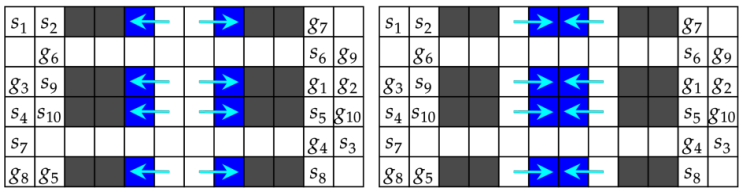
Centralized/Decentralized/Hybrid Coordination: The burden of coordination can be completely left to a single robot (the **leader**) which is referred to as **Centralized**, or shared among the robots, in a word, **Decentralized**. Some coordination processes can have both centralized *and* decentralized components, in which case they are referred to as **Hybrid**. Centralized coordination approaches can be either classified as **Strongly Centralized**, as they use a single pre-defined leader for the entire mission, or **Weakly Centralized**, where the leader may change during the mission based on some criteria (e.g. communication signal strength, available computational capability, etc.). Decentralized coordination approaches can be either classified as **Distributed**, when all robots equally share the burden of coordination, or **Hierarchical**, that is, with local leaders, for example each commanding a team of robots. Any of these coordination behaviors can be achieved regardless of communication being implicit or explicit, but not necessarily in all coordination contexts, which is why they appear twice in Fig.II.24 to convey this idea. Weak coordination is more robust to communication failure/unavailability than strong coordination, as it does not rely on an unchanging single leader, although it is more computationally demanding and complex to implement [16].

II. 2.2.2 Multi-Robot works with movable obstacles

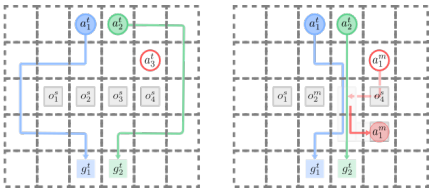
As previously presented in Section II. 1.9, Multi-Robot NAMO has not been studied yet, beyond the *heterogeneous* drone and manipulator robot system of Mueggler et al. [12]. While being a relevant use case, their system dodges the difficulty of a shared space, and the associated *explicit* and *strongly centralized* algorithm can not scale beyond the two robots, as it was never meant to deal with conflicts. Hence, in this section, similarly to Section II. 2.1, we shall present works that somewhat resemble NAMO, in multi-robot variations of the robotics problems closely related to NAMO (as a reminder: Navigation Planning, Manipulation Planning, Rearrangement Planning, Assembly Planning, and combined Task and Motion Planning).

Navigation Planning: Interestingly, multi-robot considerations, similarly to social ones, have been most studied for the Navigation Planning problem [16]. Most of the work on Multi Robot Navigation Planning can be found under the name of **Multi-Agent Path Finding** (abb. MAPF), extending the Piano Mover's problem of computing paths for each agent (i.e. robot) to their respective destinations, that can be concurrently executed while guaranteeing the absence of

a collision [106, 107]. MAPF surveys [106][105] show that until 2019, the MAPF literature has been focused on static grid environments where the agents are generally cell-sized, but also most importantly, the only entities allowed to move: in MAPF, agents can not move obstacles. The same surveys cite Conflict-Based Search (abb. CBS) [108] as the now most popular algorithm to solve MAPF. It is only very recently, in a time period (2020-2022) coincidentally overlapping with that of the present thesis, that the MAPF literature started considering the possibility of movable obstacles, proposing extensions of CBS to this context [17, 18]. Bellusci et al. [17] introduced Configurable MAPF (abb. C-MAPF), which consists in computing both an environment configuration that helps improve robot circulation and a solution to a MAPF problem in said environment configuration. The problem formulation does authorize robots to move the obstacles themselves as they execute their motion, but rather assumes that humans will arrange the environment following the solver's solution prior to having the robots navigate, so that they may circulate more efficiently (Cf. Fig.II.25a showing two environment configurations between which the solver should select the one that permits to solve the MAPF problem better). In their simulated experiments, they actually had the solver remove rather than move obstacles from the environment, further distancing the problem from NAMO, where obstacles can only be moved. In contrast, Vainshtain et al. [18]'s extension to MAPF, "Terraforming MAPF" (abb. TF-MAPF), is formulated under the hypothesis that actual agents move obstacle *during* execution. However, as shown in Fig.II.25b, while this problem is much closer to NAMO, it still differs greatly in that: only specific robots may manipulate objects (they do not have a navigation goal themselves), and they can only manipulate shelve-like objects by passing under them and lifting them.



(A) A warehouse where blue shelves can be put in two configurations **by humans before robot motion execution**, and $n = 10$ agents need to move from cell s_i to cell g_i



(B) (Left) MAPF problem: agents a_1 & a_2 must circumvent obstacles to reach goals g_1 and g_2. (Right) TF-MAPF problem: robot a_1^m passes under o_4 to lift o_3 and open a shorter path for a_1 & a_2.

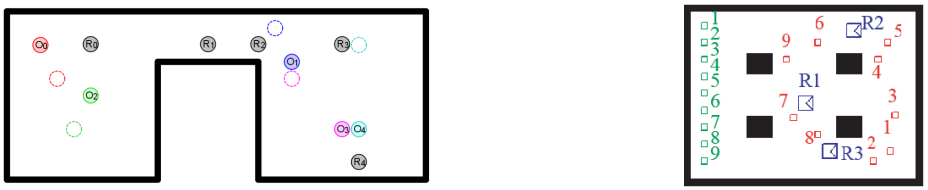
FIGURE II.25: (A) C-MAPF problem from Bellusci et al.'s paper [17], (B) MAPF and corresponding TF-MAPF problem from Vainshtain et al.'s paper [18]. Both consist in having robots navigate in a warehouse with movable shelves. Vainshtain et al.'s approach is even better understood with its accompanying video: `https://bit.ly/3ImgfAw`.

In both cases, **these approaches remain firmly limited to cell-sized robots and movable obstacles, in contrast with the evolution of NAMO towards more realistic world models shown in Section II. 1**. While MAPF is mostly studied and solved in a *centralized* fashion with *explicit* communication (CBS), the larger problem of multi-robot navigation coordination is generally approached in a *decentralized* fashion with *explicit* communication, in order to share the computational costs between the robots, according to Verma & Ranga [16]. To this date

(2023), we could not find other mentions of movable obstacle use in the larger multi-robot navigation literature.

Manipulation Planning: From a recent survey of collaborative manipulation in MRS [109], Verma & Ranga [16] drew the conclusion that most research under the umbrella of Multi-Robot Manipulation Planning focuses on the Multi-Agent Object-Pushing Problem (e.g. [110]). In this problem, robots must move a single object together to a given position, focusing on cooperative manipulation of a same obstacle, at the same time. A large portion of works in this domain again rely on *explicit* communication and *dynamic* coordination, regardless of other characteristics. Incorporating this collaborative manipulation ability of individual obstacles may be very useful in NAMO problems, especially so for larger/heavier objects, that would be either impossible or too complex to confidently move them alone. However, this ability would only be useful on specific occasions; before delving into this widely studied specific subproblem, **it appears to us that it is more pressing to address the Multi-Robot NAMO problem under its original hypothesis that a single robot may manipulate a single movable obstacle at once.**

Rearrangement/Assembly Planning: Interestingly, Levihn, with Igarashi and Stilman (the founder of the NAMO problem), explored in 2012 the problem of Multi-Robot Rearrangement/Assembly Planning [111], instead of Multi-Robot NAMO. In stark contrast with typical NAMO problems presented in Section II. 1, robots are not blocked by movable obstacles in [111], and there are no free space components to be joined. Movable obstacles all have final position constraints that do not affect the environment's topology significantly (Cf. Fig.II.26).



(A) Levihn et al.'s example environment from [111]. Robots are annotated with *R*, movable obstacles with *O*, and corresponding goal configurations in empty color-matching circles.

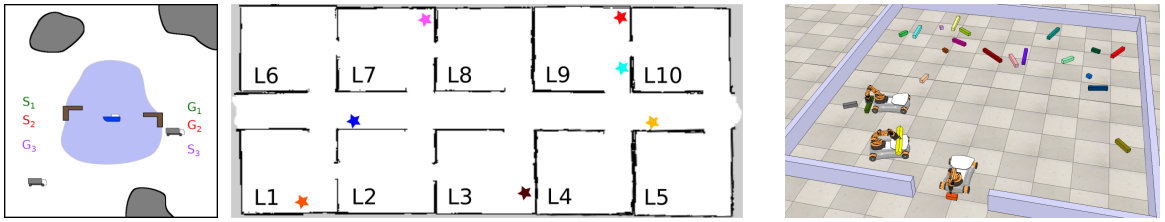
(B) Oyama et al.'s example environment from [112, 113]. Robots are annotated with *R*, initial obstacle configurations are red, goal configurations in green.

FIGURE II.26: Example environments in the Multi-Robot Rearrangement Planning literature: the initial and goal obstacle configurations do not really hamper robot motion to the point of dividing free space into separate components.

All other works we could find with multiple robots navigating in a shared plane to rearrange objects ([114, 112, 113]) were prior to Levihn et al.'s work, and shared the same aforementioned characteristics differentiating them from NAMO. These prior works and Levihn et al.'s were all *strongly centralized*, and required *explicit* communication. Later works under the name of "Multi-Robot Rearrangement Planning" focused on experiments with multi-robot arms fixed on a table [115, 116, 117], mainly consisting in Assembly Planning problems (with priority constraints in object placement to assemble the goal construct), making them much

less relevant to our subject of study. Both rearrangement planning and assembly planning arguably greatly differ from the NAMO problem, since the final position constraints they assume for each movable object drastically reduce the search space. **In none of these papers do the robots need to choose which obstacles to move and where to leave them, which is actually where a great part of the NAMO problem's difficulty lies**.

Combined Task and Motion Planning (TMP/TAMP): Finally, let us discuss Multi-Robot TMP, as TMP is a super-set of the NAMO problem and all the aforementioned related problems, according to the recent survey of Garrett et al. [57]. The same survey shows that, while problems that can be qualified as TAMP have existed for at least as long as robotics have been studied, it is only recently, in the last decade, that the term has emerged and gained traction. In the first and only survey we could find on Multi-Robot Task and Motion Planning by Antonyshyn et al. [118] (2023), we could not find Multi-Robot NAMO problems per se. In cited papers focused on 2D navigation problems associated with manipulation of objects, such as Motes et al.'s and Thomas et al.'s papers [119, 120], the objects either do not occupy physical space that would hamper the robots' navigation, or are not actually manipulated by the robot itself (Cf. Fig.II.27a and II.27b).



(A) Motes et al.'s example environment from [119].

(B) Thomas et al.'s example environment from [120].

(C) Tang et al.'s example environment from [121].

FIGURE II.27: Example environments in the Multi-Robot combined Task and Motion Planning literature. In (A), trucks and boats exchange loads that don't occupy space beyond the agent's geometrical footprint. In (B), room have doors that can be closed or opened during execution, but by "deus ex machina"-like actions where a human would instantaneously change the door's state, not by the robot. In (C) the robots must remove all obstacles from the room.

Interestingly, the survey underlines the lack of real-world experimentations and validation, similar to our own conclusion about the NAMO domain (Section II. 1.10), and in contrast to existing conclusions on the Socially-aware robot navigation, task and motion planning domain [14, 13]. It also makes the conclusion that comparatively fewer approaches are based on *decentralized* than *centralized* coordination, and further research on *decentralized* coordination would be welcome to improve the applicability of TAMP algorithms to environments with either a large number of robots, or where individual robot failure is likely, or where communication capabilities are limited. Beyond this survey, the Multi Robot combined Task and Motion Planning paper closest to the NAMO we could find was Tang et al.'s [121], discussing the problem of Multi-Robot Clutter Removal (abb. MRCR), consisting in emptying a specified space from all obstacles occupying it. While similar to NAMO in appearance, this problem assumes the

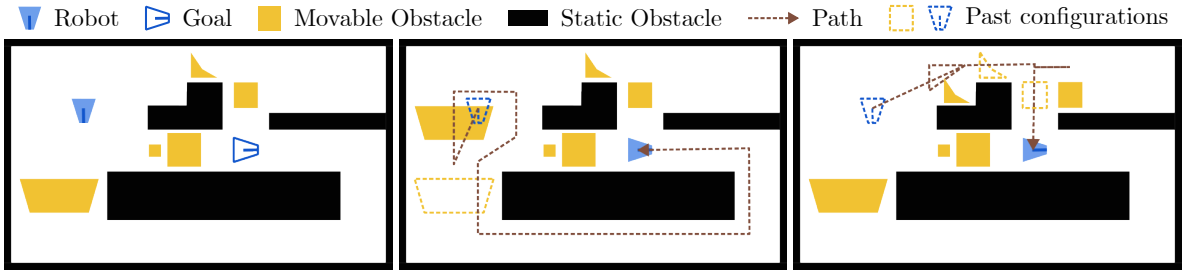
possibility of removal to infinity, and requires all objects to be moved, regardless on their impact on robot navigation efficiency (Cf. Fig.II.27c). **As underlined in the previous paragraph, choosing which obstacles to move and relevant placements are characteristic subproblems of NAMO, and need to be properly addressed**.

Conclusion: In the end, similarly to Socially-Aware concerns, it is apparent that none of the closest Multi-Robot problem formulations we could find actually address what we aim for: having multiple agents efficiently navigate in cluttered human indoor environments, where they need to also plan the manipulation of obstacles as needed to reach their goals. As such, we are free to formulate Multi-Robot NAMO as a new problem, which we do in Chapter V. For a first formulation, it would make most sense to start off with a fully *homogeneous* Multi-Robot System, as it is simpler to devise. In order to ensure maximum robustness regarding the possibility of individual robot or communication failure, we consider that the first approach to Multi-Robot NAMO should be under the assumption of a Multi-Robot System that only allows *implicit* communication, and use a fully *distributed* coordination strategy. That way, more complex future Multi-Robot NAMO systems that would rely on *explicit* communication to coordinate in more efficient manners, would always have a backup solution in case of aforementioned failure. Under this hypothesis we make of no *explicit* communication, robots would not be able to communicate their individual navigation goals to each other: as such the system we are envisioning for this thesis could be considered mostly *competitive*, although we shall see in Chapter V how we may still create *cooperative* behaviors under these assumptions.

Chapter III

Revising and simulating reference NAMO algorithms

From the previous state of the art, we have identified two relevant baseline algorithms upon which we will build social and multi-robot extensions to NAMO in the following chapters: (Wu&Levihn, 2014) [7] and (Stilman, 2005)'s [3] algorithms. In this chapter, we argue these choices and present an upgraded version of these algorithms. Indeed, we found limitations (e.g. action space restrictions, edge cases and sensing requirements) that we consequently addressed, while preserving the original theoretical properties, under a more coherent and generic formalization. All these improvements facilitate experimentation in the following chapters. These algorithms (illustrated in Fig.III.1) have been implemented, and tested, in the open simulator and datasets we created to fill the gap left by the lack of tools targeted at the study of NAMO problems. This simulator is presented in the second part of this chapter, setting the stage for our extensions to the NAMO problem in subsequent chapters.



(A) Initial world state - the robot is assumed to know all about it at t_0

(B) (Wu&Levihn, 2014)'s algorithm is only capable of moving the big trapezoid, taking the longest-distance path

(C) (Stilman, 2005)'s algorithm moves the two top-most obstacles, taking a shorter-distance path

FIGURE III.1: Example solutions of algorithms implemented in our simulator. Neither algorithm is capable of planning through the middle obstacles because manipulating the big one requires to first manipulate the small one.

III. 1 Our NAMO Problem Formalization

Before discussing our two chosen algorithms, let us first lay down some notations and vocabulary from both (Wu&Levihn, 2014) [7] and (Stilman, 2005)'s[3] algorithms, as to create a

common and coherent problem formulation for both. For this, we fuse the formalisms of Stilman's PhD thesis for Navigation Among Movable Obstacles [19], and of Levihn's Master thesis for Navigation Among Movable Obstacles in Unknown Environments [34].

III. 1.1 Workspace

The NAMO problem has been originally formulated by Stilman [19] as an instance of geometric motion planning [122]. As such, the problem can be defined in a *workspace* W (syn. *world*, or *environment*), that is modeled as a 2D or 3D Euclidean space, containing a set of distinct *entities* $E =< E_1, \ldots, E_e > | E \in W$ that are unambiguously separated into the following:

- a single *manipulator robot* $R \in E$ with n degrees of freedom,
- a set of *obstacles* (syn. *objects*) $O =< O_1, ..., O_o > \subset E$ that are either:
  - *fixed* (syn. *static*) obstacles $F =< F_1, ..., F_f > \subset O$, that cannot be moved by the robot,
  - *movable* (syn. *manipulable*) obstacles $M =< M_1, ..., M_m > \subset O$, s.t. $F \cap M = \emptyset$ that can be moved by the robot (note that $O = F \cup M$).

While the robot may be articulated (as in Stilman's original definition and some of the implementations presented in our NAMO State of the Art in Section II. 1), and thus composed of several links (syn. rigid geometries), obstacles are represented using a single rigid geometry. In the literature, such geometries are defined as polyhedrons (3D) or polygons (2D), but this is more a matter of implementation than definition. For the sake of simplicity, and without loss of generality, we assume that each entity $E_i \in E$ is a non-self-intersecting polygon in the rest of this thesis, as illustrated in Fig.III.1.

The space occupied by an entity at a time t is written E_i^t; more concretely, assuming any entity is an undeformable polygon, E_i^t designates its vertices' coordinates at time t, and the volume (i.e. the set of points) they encompass. Since the entities are assumed to be rigid bodies, an entity's *state* (syn. *configuration*) $q_{E_i}^t$ at time t can be reduced to its position and orientation; more concretely, for rigid polygons, $q_{E_i}^t = (x_{E_i}^t \in \mathbb{R}, y_{E_i}^t \in \mathbb{R}, \theta_{E_i}^t \in [0, 2\pi])$.

The NAMO problem consists in having the robot reach a *final configuration* (syn. *goal configuration*) q_R^g. This final configuration only constrains what the last robot state must be in terms of chronological order, but does not specify a specific time for it to be reached.

A *world state* (syn. *world configuration*) at time t may thus be defined as the set of entity configurations $W^t =< q_{E_1}^t, ..., q_{E_e}^t >$. Actually, since only the robot and movable obstacles may have different configurations depending on time, we can further reduce a world state to $W^t =< q_R^t, q_{M_1}^t, ..., q_{M_m}^t >$. The initial world state can be referred to as W^{t_0}. Similarly, we may refer to the state of all obstacles at time t as O^t, all movables obstacles as M^t, and all static obstacles as F^t. We write $W_{/E_i}^t$ the state of the world without considering entity E_i (this is particularly relevant to represent that intersections should be ignored with the obstacle M_i selected for manipulation during some planning computations). Similarly, we write $W_{\cup E_i}^t$ the state of the world considering an extra entity E_i.

We write as $\mathcal{C}_W$ the abstract *space of all possible configurations/states* (also referred to as *configuration space*) of W^t. The dimension of this abstract space is equal to the number of variables in W^t, multiplied by the number of admissible values they may have; concretely, assuming rigid polygons with 3 degrees of freedom, assuming n admissible values sampled for each variable, that is $n^{3\times(m+1)}$ (the number of states grows exponentially with the number of obstacles).

In Stilman's original NAMO problem formulation [19], it was assumed that not only geometric, but also kinematic and dynamic parameters of every entity were part of the world model, such as object mass, center of mass, moment of inertia, or friction parameters - as these parameters were directly available to the robot through their simulation tool. While this allows more physically realistic estimation of obstacle movement and energy consumption, it arguably greatly complexifies the model, making it that much harder to implement, write test scenarios and overall reliably experiment with. Also, in the real world, accessing such parameters for obstacles is non-trivial, even for human beings - hence we shall assume a purely geometric world model in this thesis, a choice shared by Wu&Levihn's model.

III. 1.2 Actions and Action spaces

III. 1.2.1 Actions

Any state "changes", that is, transitions between world states, are expressed as *actions* (syn. *operators*) between time steps t and $t+1$, written as $A(t, t+1)$ or A^t for short. Actions describe continuous motion between these time steps: an action can thus be interpreted as following a path or trajectory $\tau(q_R^t, q_R^{t+1})$[1]. Following this path implies covering a *swept volume* (eq. *swept area* in 2D) $S(q_R^t, q_R^{t+1}) = \bigcup_{t_a \in [t,t+1]} R^{t_a}$, corresponding to all point in space that are occupied by the robot within t and $t+1$. $S(A^t)$ can be used as an abstract notation of the swept volume for any action A^t. NAMO distinguishes between two primitive (voluntarily abstract) operators/actions:

- *Navigate*, which refers to the contact-free *transit* paths of defined by Alami et al. [40] we mentioned in Section II. 1.3. More precisely, the robot may be in sliding contact with obstacles, but its motion must not displace any obstacles:

$$\begin{aligned} &Navigate : (W^t, \tau(q_R^t, q_R^{t+1})) \rightarrow W^{t+1}, \\ &\text{s.t. } S(q_R^t, q_R^{t+1}) \cap O^t = \emptyset \end{aligned}$$

- *Manipulate*, which refers to the manipulation of an object O_i, or its *transfer* [40], again without collision of neither robot nor manipulated obstacle with other obstacles. Naturally, in addition to the robot's path $\tau(q_R^t, q_R^{t+1})$, the *Manipulate* operator also implies a path $\tau(q_{M_i}^t, q_{M_i}^{t+1})$, covering a swept volume $S(q_{M_i}^t, q_{M_i}^{t+1}) = \bigcup_{t_a \in [t,t+1]} M_i^{t_a}$. The operator is however not directly parameterized by the manipulated obstacle's path: instead it is parameterized by the *initial contact* (syn. grasp) $G_j \in G(M_i)$, $G(M_i)$ being the set of relative

[1] It is important to note here that while paths may not be explicitly parameterized by time, we use the variable t to refer to a chronological ordering on states and operations.

transformation functions, mapping and constraining the obstacle's motion to the robot's, where grasping the obstacle is possible through contact. Distinct initial contacts or grasps G_j yield distinct obstacle motion given the same robot trajectory $\tau(q_R^t, q_R^{t+1})$:

$$Manipulate : (W^t, O_i, G_j, \tau(q_R^t, q_R^{t+1})) \rightarrow W^{t+1},$$
$$\text{s.t. } S(q_R^t, q_R^{t+1}) \cap O^t = \emptyset \text{ and } S(q_{M_i}^t, q_{M_i}^{t+1}) \cap O^t / M_i = \emptyset$$

Action and state validity not only depend on the absence of collision with other objects, but also on actual robot capabilities. The exact formulation (also called *action space* $\mathcal{A}$) of *Navigate* and *Manipulate* abstract actions depends on implementation, which we shall discuss in Section III. 3.2. Instantiated actions of the *Navigate* abstract action are denoted as $A_N \in \mathcal{A}_N$, and as $A_M \in \mathcal{A}_M$ for *Manipulate*.

III. 1.2.2 Action sequences (Plans)

A *Plan* (syn. *Action Sequence*), denoted $\mathcal{P}$, is a continuous succession of actions indexed from A^{t_s} to A^{t_e}, where the resulting configuration of affected entities for one action is equal to the starting configuration of the same entities for the next action. The way actions have been previously defined, a plan, or a subsequence of a plan (also called a plan *component*), consisting of only continuous successive *Navigate* actions can be reduced to a single *Navigate* action. Similarly, this is also true for continuous successive *Manipulate* actions under the condition that they are applied on the same movable obstacle M_i, with the same grasp G_j.

The swept volume of the entire plan can be defined as the union of the swept volumes of its individual actions:

$$S(P) = \bigcup_{t \in [t_s, t_e]} S(A^t)$$

A plan can only be valid as long as its swept volume $S(P)$ does not intersect with other obstacles than the ones being manipulated. The set of all valid plans is written $P(t)$. In this set, the subset of plans consisting purely of *Navigate* actions is noted $P_{Nav}(t)$.

III. 1.2.3 Manipulation Action Spaces

Stilman distinguished three main abstract action spaces, also called *manipulation classes*, for the *Manipulate* operator, derived from the robot grasping literature. Let us provide a summary of said manipulation classes, with a summary of their respective advantages and disadvantages, in order of increasing generality and decreasing reliability:

- **Grasping (Constrained Contact):** This refers to a rigid grasp of the object by the robot, fully constraining its degrees of freedom, creating a fixed transform relative to the robot's grasping link. This manipulation class has minimal modeling and computational requirements, and only requires an accurate geometric model to ensure reliable manipulation execution. However, this class lacks generalizability, as it typically requires the contact between the robot and the object to satisfy "form closure", in other words, that any robot's

motion should not cause involuntary object release [123]. Depending on robot capabilities, some objects such as large boxes without handles may to be too difficult or impossible to grasp.

- **Pushing (Constrained Motion):** This refers to *non-prehensile* [123] manipulation, which can only partially constrain the object's degrees of freedom. By restricting the robot's motion instead of the obstacle's, it is possible to guarantee a fixed transform relative to the robot's pushing link and the obstacle, thanks to the static friction that prevents the objects from unexpectedly slipping [124, 125]. While more general than grasping in terms of number of manipulable obstacles, it either requires extra modeling or additional hypotheses as to friction properties. Any deviation from the friction model may cause unreliable plan execution because of obstacle slip.
- **Manipulation Primitives:** Refers to the use of forward physics simulation to predict the transformation between robot and object motion, for both grasping/pushing and other modes of interaction that allow translational or rotational slip [126, 127], with various degrees of constraint in both grasp and motion. This allows for non-fixed transforms relative to the robot, yielding motion solutions that the other two manipulation classes can not. This generality comes at the cost of even more modeling of dynamic properties of the obstacle (e.g. mass, center of mass, moment of inertia, ...), which leaves even more space for unreliable plan execution caused by any errors in estimating these parameters.

In conclusion, the more constraint in contact or motion, the fewer parameters need to be modeled, and the more reliable the plan execution is, at the cost of generality. As stated earlier, the choice of a manipulation class, or combination thereof, depends on actual robot's capabilities and the actual environment being modeled. For instance, Stilman used Manipulation Primitives in their algorithm's implementation [3], while Wu & Levihn first used Pushing then Grasping in their algorithm's implementations [33, 7]. We shall discuss our specific choice of implementation later in Section III. 3.2.

III. 1.3 Sensing

In Stilman's original NAMO problem formulation [19], it was assumed that the geometric, kinematic and dynamic parameters of every entity were known to the robot, without uncertainty in sensing or as to the effect of robot actions on these parameters, at any time. Basically, the robot was assumed to be *omniscient*, meaning there was no difference to be made between the robot's world model, and the actual world (real or simulated) - a very strong, though convenient assumption.

Wu&Levihn's problem formulation extended NAMO to Unknown Environments, and by doing so, created a need to differentiate between the *actual world* (syn. *reference world*) where actions are executed, which we shall refer to as W_{ref}, and the *world as perceived by the robot* W_R. In their formulation, it is assumed that the robot senses its environment in between each action execution, collecting partial information as to obstacles' geometry only, resulting in three possible types of incoherence between W_{ref} and W_R at any time t:

- The sets of static or movable obstacles $F, M \subset W_R$ are erroneous: the robot may have misclassified an obstacle as movable or static, by assuming its movability before interaction.
- The sets of obstacles $O, F, M \subset W_R$ are incomplete: in other words, the robot may have not detected all obstacles yet,
- The geometry model of an obstacle $O_i^t \subset W_R$ is incomplete: the obstacle may not have been fully observed yet.

This new formulation requires two extra hypotheses in order to handle these possible inconsistencies:

- **Obstacles are assumed movable by default:** Detected obstacles belong to $M \subset W_R$ unless a *Manipulate* operator has failed (eq. the resulting manipulated obstacle's state is not as expected). This allows to handle erroneous classification.
- **Unknown space is free space:** Space that has yet to be observed is assumed to be free. This allows to handle both incompleteness of obstacle sets and geometry models.

Under these same sensing hypotheses, it is possible for Stilman's algorithm (or any other existing offline NAMO planning algorithm, for that matter) to be applied to unknown environments too, as long as its plans are recomputed when newly available data becomes available. This may however be computationally costly, as Stilman's algorithm explores a much greater part of the search space, and also because the algorithm is not *dynamic*, in that it does not reuse previously computed plans to hasten new computations.

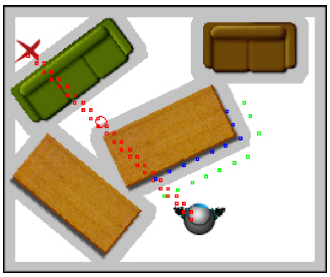
III. 1.4 Domain Restrictions

Stilman's and Wu&Levihn's formulations share some additional restrictions, beyond the elements that have been defined so far:

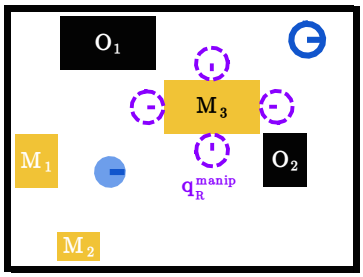
One obstacle at a time The robot may only manipulate a single obstacle at once, which may not interact with other obstacles during this process (in other words, a second or more obstacle(s) cannot be pushed through contact with the manipulated obstacle). This hypothesis shall be preserved, motivated by the previously made observation that our state of the art (Cf. Section II. 1) shows that the entire NAMO literature operates under this hypothesis (except for the "pre-NAMO" era paper of Chen&Hwang [38]).

Grasping configurations The robot may only grasp the obstacle at specific sampled *grasping points* along the obstacle's edges, requiring the robot to be in specific *grasping configurations* to reach them. More precisely, Stilman's formalization assumed an even distribution of the grasping points (with an arbitrary spacing distance), and derived one robot grasping configuration for each at a radius distance from the grasping point (Cf. Fig.III.2a). Wu&Levihn's formalization reduced Stilman's set of valid grasping points to the center of the axis-parallel sides (Cf. Fig.III.2b). On one hand, this choice significantly reduces the search space and makes it

more independent of obstacle size (at most 4 grasping points instead of as many as one can fit on the obstacle's sides at a fixed distance), making it less computationally costly. On the other hand, it correspondingly reduces the solution space, constraining the number of solvable problems (i.e. generality). This reduction however physically makes sense, as it is far easier to constrain an object's movement by grabbing it closest to its geometric center (which we as humans naturally do). **In this thesis, we shall restrict grasping points similarly to Wu&Levihn, but to the center of all obstacle sides instead of only axis-aligned sides, as to not be restricted to an axis-aligned world model**. In any case, our new algorithmic formulations for both algorithms allow for any method of computation of valid grasping configurations, as long as it yields a finite set of such configurations.



(A) In blue, valid grasping points sampled along the table's edge, in green, corresponding robot configurations, in (Stilman, 2005)'s formulation.



(B) In purple, valid robot grasping configurations for obstacle M_3, for (Wu&Levihn, 2014)'s formulation.

FIGURE III.2: Illustration of the different grasping configuration restrictions for (Stilman, 2005) and (Wu&Levihn, 2014)'s problem formulations

Goal is always free Both Stilman's and Wu&Levihn's formulations (and subsequent algorithms) suppose that the robot goal configuration q_R^g must remain free of obstacles for the entire plan's duration. In the following section, we generalize their two algorithms to remove this constraint when a single movable obstacle M_i intersects with q_R^g.

Action space restrictions Wu&Levihn's formulation only allowed axis-aligned translational manipulations of obstacles. In the following section, we show that this restriction is not actually a fundamental requirement of the problem definition, and can be dropped by generalizing the algorithm to a continuous world model instead of a grid-discretization of it.

III. 1.5 Cost & Optimality

As mentioned in Chapter II, Navigation Among Movable Obstacles being fundamentally a Navigation problem, it seeks to minimize the robot *displacement cost* associated with each action A^t, of an estimation function $\mathcal{C}(A^t)$ of distance, time or energy, depending on the implementation hypotheses as to the robot capabilities and the physical characteristics of obstacles. As such, we can write the cost of a plan $\mathcal{P}$ as the cumulated sum of each of its actions:

$$\mathcal{C}(\mathcal{P}) = \sum_{t \in [t_s, t_e]} \mathcal{C}(A^t)$$

The exact computation method of this cost is mainly implementation-dependent, according to what the user wishes to optimize. Still, as we've noted in our state of the art (Cf. Section II. 1.10, it is relevant in NAMO to differentiate the cost of *Navigate* and *Manipulate* operators, generally through an approximation of *energy* (syn. *work*), to account for the additional difficulty of displacing an obstacle in addition to the robot's body.

On one hand, since Stilman's omniscient world and sensing model assumed knowledge of kinematic and dynamic parameters in addition to geometry, for every entity in the world, the work cost could be trivially accurately estimated as the product of Force and Traversed Distance for both the robot and the obstacle. On the other hand, since Wu&Levihn's world and sensing model did not offer kinodynamic data, they chose to operate with a simplified model of weighted distance similar to that of Chen&Hwang [38], assuming a constant force estimate for *Navigate* and *Manipulate* operators, respectively written $\mathcal{C}_N$ and $\mathcal{C}_M$ under the constraint that:

$$\mathcal{C}_M > \mathcal{C}_N \geq 0$$

As Wu&Levihn's problem formulation also assumed uniform action cost as actions were limited to axis-aligned unit translations, assuming a number of n navigation actions and m manipulation actions, the plan cost could easily be computed as:

$$\mathcal{C}(\mathcal{P}) = n \times \mathcal{C}_N + m \times \mathcal{C}_M$$

Given that, as argumented in Section III. 1.3, we opted for the fully geometric world representation of Wu&Levihn, we also opted for their *constant-force* cost model, albeit generalized to allow rotations and non-uniform costs (for instance, to properly reflect the additional energy consumption of a diagonal movement in a square grid). In this generalized model, two additional *constant moment* estimates for *Navigate* and *Manipulate* actions are needed, respectively C_N^θ and C_M^θ, to weigh the angular distance due to rotational movement. Similarly, the force constants C_N and C_M are used to weigh the euclidean distance due to translational movement. These two products yield a generic energy cost estimate that can be added to obtain the actual action cost for *Navigate* and *Manipulate* actions respectively[2]:

$$\mathcal{C}(A_N(t, t+1)) = C_N \times \sqrt{(x_R^{t+1} - x_R^t)^2 + (y_R^{t+1} - y_R^t)^2} + C_N^\theta \times (\theta_R^{t+1} - \theta_R^t)$$

$$\mathcal{C}(A_M(t, t+1)) = C_M \times \sqrt{(x_R^{t+1} - x_R^t)^2 + (y_R^{t+1} - y_R^t)^2} + C_M^\theta \times (\theta_R^{t+1} - \theta_R^t)$$

We note as $\mathcal{P}^*$ the plan with minimal cost:

$$\mathcal{P}^* = argmin_{\mathcal{P} \in P(t)}(\mathcal{C}(\mathcal{P}))$$

[2]More concretely, in all our experiments, we choose parameter values so that the cost of a unit rotation action is equal to the cost of a unit translation action in axis-aligned directions, and manipulations actions are twice the cost of equivalent navigation actions.

III. 1.6 Space Components and Openings

III. 1.6.1 Space Components

In Section III. 1.1, we defined the configuration space $\mathcal{C}_W$ as the abstract *space of all possible configurations/states* of the world W^t. We write the *slice* (syn. *subspace* or *space component*) of all robot configurations $C_R(W^t)$. This subspace can be further cut into more specific spaces that will extensively be used in the next sections:

- the space of all collision-free robot configurations:

$$C_R^{free}(W^t) = \{q_R \in C_R(W^t) | R \cap O^t = \emptyset\}$$

- the space of all *accessible* robot configurations, more precisely, all the collision-free robot configurations that can be reached from the current robot configuration using only *Navigate* actions:

$$C_R^{acc}(W^t) = \{q_R \in C_R^{free}(W^t) | \exists \tau(q_R^t, q_R) \text{ s.t. } S(q_R^t, q_R) \cap O^t = \emptyset\}$$

- the space of all colliding robot configurations with a movable obstacle M_i, noted:

$$\chi_R^{M_i} = \{q_R \in C_R(W^t) | R \cap M_i \neq \emptyset\}$$

- the space of all robot configurations that **exclusively** collide with a specific movable obstacle, noted:

$$\textbf{exc}\ \chi_R^{M_i} = \{q_R \in \chi_R^{M_i}(W^t) | R \cap O^t_{/M_i} = \emptyset\}$$

The space of all collision-free robot configurations, $C_R^{free}(W^t)$, can also be expressed as the union of all *free space components*, the disjoint sets of robot configurations where the robot may *Navigate* from any one configuration to another without ever having to use a *Manipulate* operator:

$$C_R^{free}(W^t) = \{C_1, \dots C_d\} \text{ with } C_i = \{q_R \in C_R^{free}(W^t) | \exists \tau(q_R^1, q_R^2) \text{ s.t. } S(q_R^1, q_R^2) \cap O^t = \emptyset\}$$

III. 1.6.2 Openings

Another fundamental notion to Stilman and Wu&Levihn's formulations (and arguably, to the whole NAMO domain), is the concept of *new opening*. When a robot moves obstacles in a NAMO problem, it is for the purpose of finding a lower-cost path to its goal configuration - or finding a path at all. When the robot considers possible motions for an obstacle, it must thus determine when a new - ideally lower-cost - path to the goal has been indeed been created by a specific motion sequence, as to decide whether it is worth considering or not: this is, in broad words, checking for the existence of a new opening. Let us now formally define this concept of new opening, using the notion of *homotopic* plans, that needs to be defined first, derived from the definition of Igarashi & Stilman [128]:

Homotopic plan: Two pure navigation plans $P_1, P_2 \in P_{Nav}(t)$ are homotopic if and only if there exists no known obstacle in the area enclosed by the paths that the robot traverses if executing the plans. Otherwise, they are ahomotopic.

New opening: A *Manipulate* action created a new opening if the set $P_{Nav}(t')$ at time t' after the execution of the *Manipulate* action has at least one plan P that is ahomotopic to all the plans in the set $P_{Nav}(t)$ at time t prior to the execution of the manipulation action.

III. 1.7 NAMO Problems

Having properly formalized the context, we can formulate a generic definition of the NAMO problem:

General NAMO Problem: Given an initial workspace configuration W^{t_0}, the NAMO problem consists in computing, if it exists, a plan $\mathcal{P}$ for a manipulator robot R from its initial configuration $q_R^{t_0}$ to a goal configuration q_R^g, while being allowed to *Manipulate* movable obstacles $M_i \in M$ as necessary, guaranteeing the absence of any collisions between the robot R and manipulated obstacles with any other obstacles $O_i \in O$.

Optimal NAMO Problem The optimal NAMO problem further constrains the General NAMO Problem to computing the minimum-cost plan $\mathcal{P}^*$ under the same otherwise stated constraints.

III. 2 Baseline NAMO Algorithms

In this section, we first present and illustrate the original algorithms' logic, then provide and explain our improved pseudocode formalization. The original pseudocode formalization of these algorithms is provided in Appendices B, C and D. Hence, if you wish to have a better understanding of the originals, we invite you to keep the Appendices pages at hand while reading this chapter for easier reference - this is however not necessary if you only wish to understand the algorithms themselves. Still, this section does assume a basic understanding of standard graph search algorithms (Breadth-First Search, Dijkstra and A^*). Appendix A conveniently provides a reminder about these algorithms - and, more importantly, a pseudocode formalization that fits with the one used in this entire document. If you are unfamiliar with these algorithms, please do consult this appendix first before continuing.

III. 2.1 (Wu&Levihn, 2014)'s Algorithm

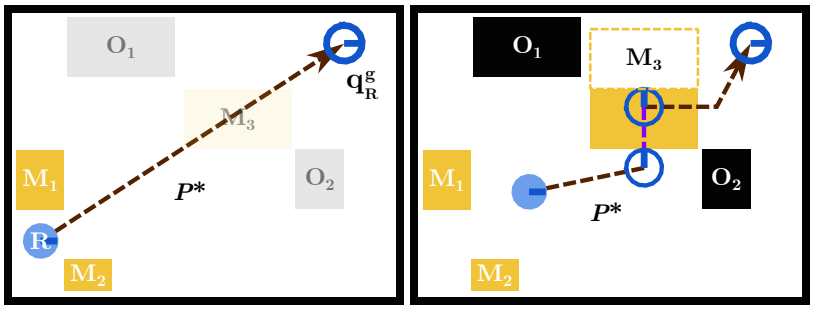
One of the original ambitions of our work as it started in 2018, was to execute NAMO (with our social & multi-robot extensions or not) on a humanoid Pepper robot, using only its onboard sensors, with only a map of static obstacles (like walls and such) as prior knowledge. Hence, our attention quickly turned to the corpus of work of Wu&Levihn [33, 34, 35, 7], as it met this requirement of limited prior environment knowledge (Cf. Table II.1), while providing a local optimality guarantee.

The other great advantage of this algorithm, is that one could arguably call it the "simplest" or "most straightforward" NAMO algorithm out there. Indeed, as it only generates plans where a **single** obstacle will be manipulated at most, it completely ignores the fundamental NAMO challenge of object manipulation ordering in order to focus on the object selection. By focusing on this single problem, the algorithm drastically reduces the search space, resulting in computationally-efficient planning in an environment that is discovered as the robot navigates. It also makes it a very relevant entry point to understand more complex NAMO algorithms, like (Stilman, 2005)'s algorithm we will present in Section III. 2.2.

III. 2.1.1 Algorithm outline

In this section, we present the original structure of (Wu&Levihn, 2014)'s algorithm. Our improvements will be subsequently highlighted in the next section, facilitating their understanding since the algorithm will have been thoroughly presented by then.

(Wu&Levihn, 2014)'s algorithm computes and executes locally optimal plans $\mathcal{P}^*$ from a robot start configuration q_R^t to a goal configuration q_R^g (Cf. Fig.III.3a). A plan is a sequence of discrete actions, that are either part of the robot's manipulation action space $\mathcal{A}_M$, or its navigation action space $\mathcal{A}_N$, depending on whether the robot is manipulating an obstacle or not. A function $\mathcal{C}$ maps each action to a positive displacement cost that approximates the real distance, time or energy expense, depending on the application. When used on a plan as $\mathcal{C}(\mathcal{P})$, the function is to be understood as the sum of the individual action costs - an optimal plan minimizes this cost function.



(A) Perceived world W^{t_0} - $O_{new} = \{M_1, M_2\}$

(B) Perceived world W^{t_1} - $O_{new} = \{O_1, O_2, M_3\}$

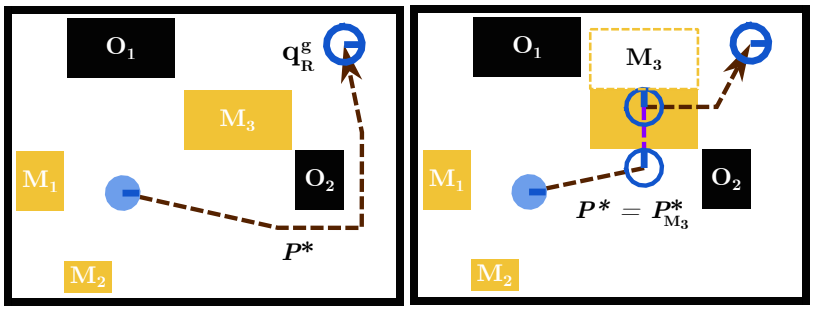
FIGURE III.3: Illustration of obstacle discovery and subsequent re-planning. Robot field of view voluntarily not shown for the sake of simplicity - it is assumed that O_1, O_2 and M_3 are suddenly discovered at t_1.

Plan execution (Wu&Levihn, 2014)'s algorithm is meant to be used with a limited robot perception - its plan changes as new obstacles are discovered. As an online planning algorithm, its formulation thus not only covers plan computation, but also its execution - checking for plan validity at each time step and triggering re-computation when necessary. As **the algorithm supposes that the environment does not change beyond the robot's own actions**, it

only needs to re-plan if the currently executed optimal plan $\mathcal{P}^*$'s remaining actions intersects with the set O_{new} of newly discovered or updated obstacles. This re-planning process is illustrated in Fig.III.3: the robot initially executes a purely navigational plan, since M_1 & M_2 do not block the way (Fig.III.3a). Upon discovering obstacles O_1, O_2 & M_3, the robot updates its plan to manipulate M_3 and reach the goal.

Task-level planner As we explained in Chapter II, NAMO algorithms are generally hierarchized with a task-level planning routine at the top, that interleaves obstacle selection and motion-level planning routines calls, in order to create the NAMO plan. (Wu&Levihn, 2014)'s algorithm follows this same structure. For this algorithm, the high-level task-planning routine consists in:

- Trying to compute an optimal navigation path $\mathcal{P}^*$ from its current configuration q_R^t to its goal configuration q_R^g, while avoiding all obstacles (Fig. III.4a),
- Then, iterating over movable obstacles, trying to compute a lower-cost **three-step plan** $\mathcal{P}^*_{M_i}$ to replace $\mathcal{P}^*$ consisting in the manipulation of a **single** obstacle M_i (Fig.III.4b). We will refer to this process as "**evaluating**" obstacles.



(A) First, the optimal plan avoiding all obstacles is computed, $\mathcal{C}(\mathcal{P}^*) = 16$

(B) Then, the optimal plan manipulating M_3 is computed, $\mathcal{C}(\mathcal{P}^*_{M_3}) = 12$

FIGURE III.4: The task planner steps that yield the plan seen in Fig.III.3b at the same W^{t_1} world state. $\mathcal{P}^*$ is replaced by $\mathcal{P}^*_{M_3}$ as it has a lower cost.

The three-step NAMO plan Before we delve into the obstacle selection routine, we need to further characterize the three-step NAMO plan. This plan always follows the same structure (illustrated in Fig.III.5):

- **Step 1**: navigation/transit path $\mathcal{P}_{to\_obstacle}$ from current robot configuration q_R^t, to a configuration near the obstacle q_R^{manip} that allows interaction,
- **Step 2**: manipulation/transfer path $\mathcal{P}_{manip}$ from the previous configuration q_R^{manip} to the configuration at which the robot leaves the obstacle q_R^{cur} so that it creates a new opening (Cf. Section III. 1.6.2),
- **Step 3**: navigation/transit path $\mathcal{P}_{to\_goal}$ from the previous configuration q_R^{cur} at which the robot leaves the obstacle, to the robot goal configuration q_R^g.

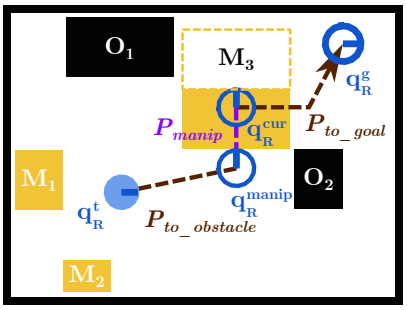


FIGURE III.5: Obstacle manipulation three-step NAMO plan - detailed notations

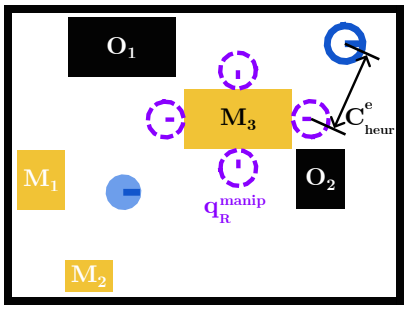


FIGURE III.6: VALID-GRASPS(M_3) in purple dots, euclidean distance cost underestimate $\mathcal{C}^e_{heur}$

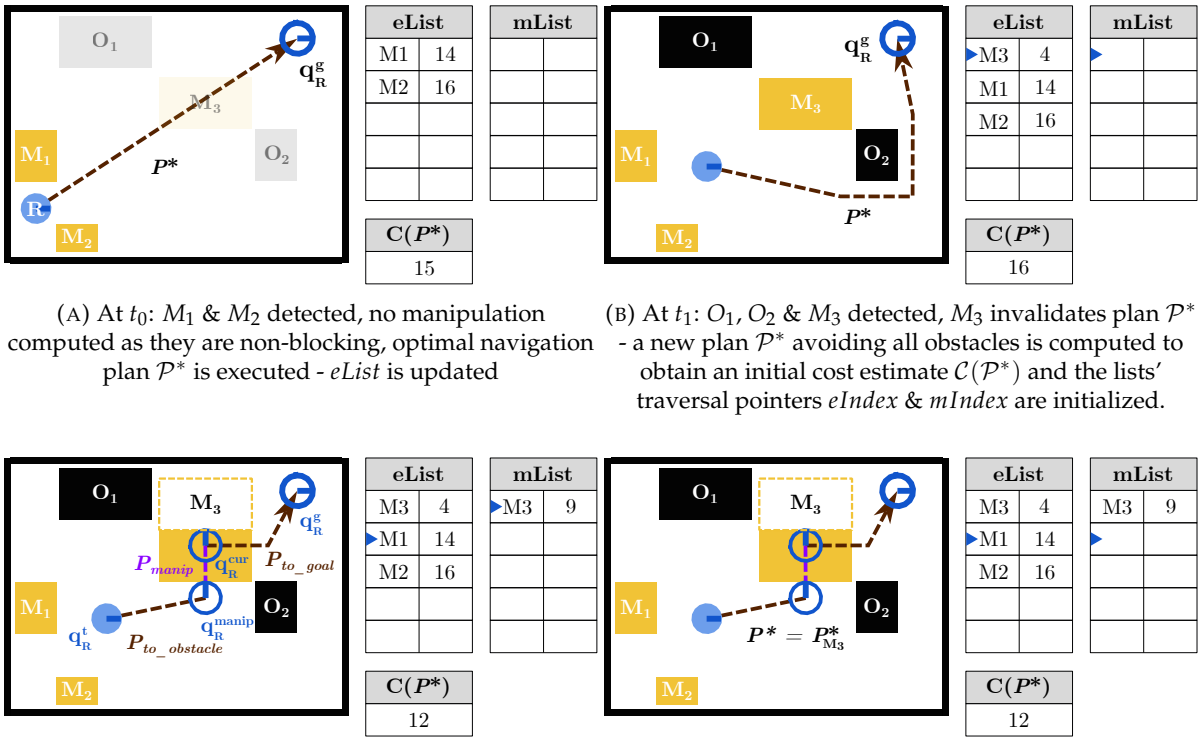
Obstacle selection We previously mentioned that the task planner iterates over movable obstacles to find a lower cost NAMO plan than avoiding all obstacles. However, so far, we have only shown the three-step NAMO plan for M_3 - not for M_1 nor M_2. These obstacles are actually never evaluated by the algorithm, as they cannot yield a better plan than M_3. This is because the obstacle selection routine prioritizes the evaluation of movable obstacles that are most likely to yield a lower-cost plan.

This prioritization of obstacles to evaluate is done through *two* ordered lists of obstacle and cost underestimate tuples (O_i, $\mathcal{C}_{heur}$): ***eList*** and ***mList*** (Represented as tables in Fig.III.7, a figure detailing the step by step resolution of a NAMO problem by the algorithm). Each list uses a different cost underestimate $\mathcal{C}_{heur}$ of the cost of the three-step plan that would manipulate the associated obstacle. For *eList*, the underestimate is the euclidean distance between the goal and its nearest manipulation configuration $\mathcal{C}^e_{heur} = min(|q^g_R - q^{manip}_R|, \forall q^{manip}_R \in$ VALID-GRASPS(M_i)[3]) (Cf. Fig.III.6). This underestimate is always valid, as it is fully updated at each re-planning. For *mList*, the underestimate is the sum of the transfer path cost and the transit path cost to the goal $\mathcal{C}_{heur} = \mathcal{C}(\mathcal{P}_{manip}) + \mathcal{C}(\mathcal{P}_{to\_goal})$, only updated when the obstacle has been evaluated (Fig.III.7c)[4]. When available, the underestimate in *mList* is always prioritized, as it is more informed. The plan search is exited early if the best underestimate for the currently considered obstacle exceeds the cost of the best plan computed so far $\mathcal{C}(\mathcal{P}^*)$ - hence why M_1 and M_2 are not evaluated. Otherwise, the plan search is stopped when all obstacles have been evaluated. This process is illustrated and explained step-by-step in Fig.III.7.

Obstacle manipulation planning So far, we have presented the high-level task planning routine and the obstacle selection routine, but we have yet to explain the three-step plan motion-planning routine itself ($\mathcal{P}_{to\_obstacle}$, $\mathcal{P}_{manip}$, $\mathcal{P}_{to\_goal}$). Once an obstacle is selected for evaluation by the task planner, the motion planning routine **iterates over the valid grasp robot configurations** $q^{manip}_R \in$ VALID-GRASPS(M_i) (Cf. Fig.III.6), to try and compute a three-step plan for each (Fig.III.8 provides a step-by-step overview of the process for one). Only if a transit path $\mathcal{P}_{to\_obstacle}$ to the grasp robot configuration q^{manip}_R can be found (Fig.III.8b), does the algorithm

[3]The valid grasp robot configurations are implementation dependent. In our case, these configurations are the ones where the robot's front's midpoint coincides with the obstacle's sides midpoints (Cf. Fig.III.6).

[4]This underestimate is only proven valid as long as no obstacle is moved - hence the reset of *mList* upon a manipulation action.



(A) At t_0: M_1 & M_2 detected, no manipulation computed as they are non-blocking, optimal navigation plan $\mathcal{P}^*$ is executed - *eList* is updated

(B) At t_1: O_1, O_2 & M_3 detected, M_3 invalidates plan $\mathcal{P}^*$ - a new plan $\mathcal{P}^*$ avoiding all obstacles is computed to obtain an initial cost estimate $\mathcal{C}(\mathcal{P}^*)$ and the lists' traversal pointers *eIndex* & *mIndex* are initialized.

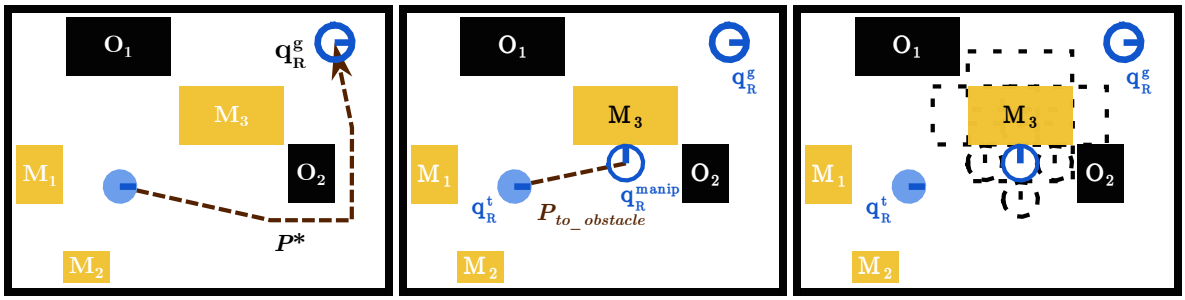
(C) Still at t_1: M_3 is evaluated because smallest entry of both lists and not yet in *mList*. Once evaluation is done, *mList* and the upper bound $\mathcal{C}(\mathcal{P}^*)$ are updated.

(D) Still at t_1: Optimal plan $\mathcal{P}^*$ is set to the three-step manipulation plan $\mathcal{P}^*_{M_3}$, as M_3 is skipped in *mList* as already evaluated, and now smallest entry in *eList* is M_1 with cost 14 > 12 (cost of $\mathcal{P}^*$). Evaluation is thus stopped, and M_2 & M_1 are not evaluated.

FIGURE III.7: Step-by-step resolution of a NAMO problem by (Wu&Levihn, 2014)'s algorithm. The blue arrows in the tables represent the traversal indexes *eIndex* and *mIndex* for *eList* and *mList* respectively.

attempt to compute, for this grasp robot configuration q_R^{manip}, the other two plan steps $\mathcal{P}_{manip}$ & $\mathcal{P}_{to\_goal}$. While $\mathcal{P}_{to\_obstacle}$ is computed by using a simple A^* graph search over the robot's navigation action space $\mathcal{A}_N$, $\mathcal{P}_{manip}$ & $\mathcal{P}_{to\_goal}$ are computed by means of a special breadth-first search we discuss below.

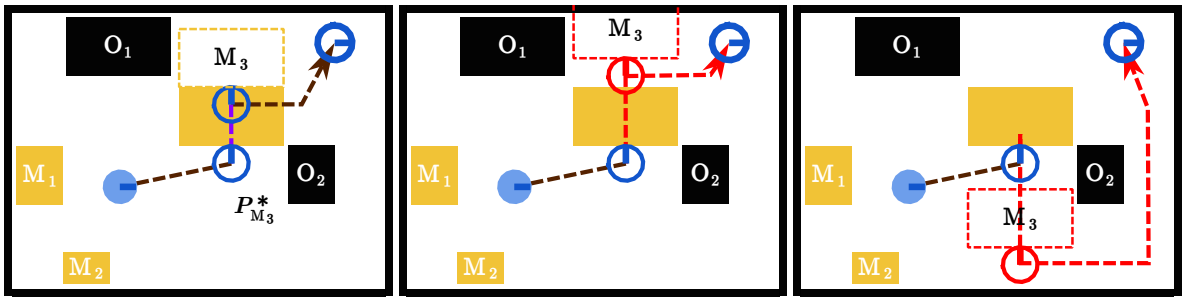
This breadth-first search explores the robot's manipulation action space $\mathcal{A}_M$. Starting from the grasp robot configuration q_R^{manip}, **actions $A_M \in \mathcal{A}_M$ are simulated to yield neighboring robot-obstacle configurations (q_R^{next}, $q_{M_i}^{next}$) - this tuple constitute the search tree node** (Fig.III.8c). Within this breadth-first search of the robot's manipulation action space, an A^* search over the robot navigation action space $\mathcal{A}_N$ is used to compute the third step $\mathcal{P}_{to\_goal}$. If $\mathcal{P}_{to\_goal}$ exists, then a global opening to the goal exists (Cf. definition in Section III. 1.6.2), and the full three-step obstacle manipulation plan, $\mathcal{P}^*_{M_i}$, can be updated (Fig.III.8d). However, this costly A^* search is not executed for every simulated action of the encompassing breadth-first search: a preemptive local opening detection subroutine (described further below) ensures this. Now is a good opportunity to remind the reader, that when the full three-step obstacle manipulation plan $\mathcal{P}^*_{M_i}$ gets computed, the associated plan cost underestimate used by *mList* (of the obstacle choice evaluation routine) is updated too.



(A) For reference: the optimal plan $\mathcal{P}^*$ avoiding all obstacles, $\mathcal{C}(\mathcal{P}^*) = 16$

(B) The transit path $\mathcal{P}_{to\_obstacle}$ to the first grasp robot configuration q_R^{manip}

(C) Breadth-first search start: valid neighbors configurations of q_R^{manip} are computed by iterating over actions $A_M \in \mathcal{A}_\mathcal{M}$



(D) New local opening detected: third plan step $\mathcal{P}_{to\_goal}$ is computed, providing a first full obstacle manipulation plan $\mathcal{P}^*_{M_i}$

(E) Collision detected: search tree branch is cut

(F) $\mathcal{C}_{est} > \mathcal{C}(\mathcal{P}^*_{M_i})$: search tree branch is cut and search is stopped, $\mathcal{P}^*$ is set to $\mathcal{P}^*_{M_i}$ as $\mathcal{C}(\mathcal{P}^*_{M_i}) < \mathcal{C}(\mathcal{P}^*)$

FIGURE III.8: Step by step manipulation planning for movable obstacle M_3, computed at t_1. Manipulation action space $\mathcal{A}_M$ simplified to axis-aligned pushes for the sake of readability.

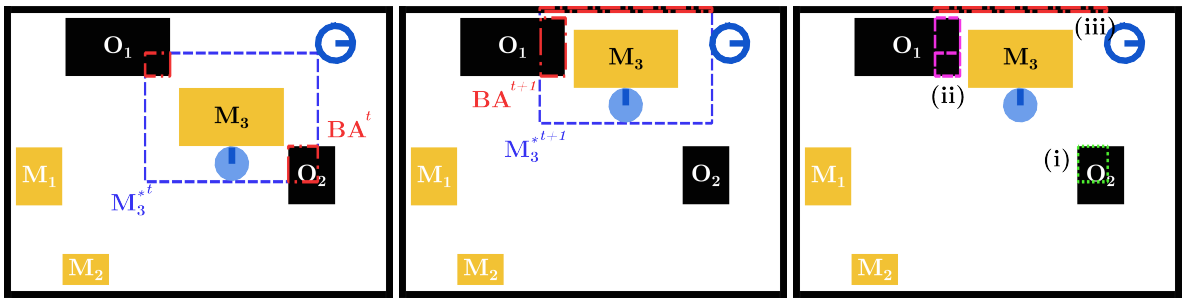
But before being added to the breadth-first search tree, neighboring configurations must meet several conditions: The most significant one - but also the most computationally intensive, hence executed last - is the collision check[5] between the next robot-obstacle state and the rest of the world representation $W^t_{/M^i,R}$. While this first condition is a fundamental constraint, the other two are computational performance improvements. The first verifies that the action does not reverse-traverse the search tree early, to avoid unnecessary costly collision checks. The second is more specific to this algorithm, and verifies whether pursuing evaluation of the neighbor (and consequentially - its successors) can actually lead to a better plan. This is achieved by underestimating the cost of the third plan step $\mathcal{P}_{to\_goal}$ as the euclidean distance $|q_R^g - q_R^{next}|$ between this neighbor configuration q_R^{next} and the goal q_R^g. If this underestimated plan cost $\mathcal{C}_{est} = \mathcal{C}(\mathcal{P}_{to\_obstacle}) + \mathcal{C}(\mathcal{P}_{manip}) + |q_R^g - q_R^{next}|$ surpasses the cost of the current best plan $\mathcal{C}(\mathcal{P}^*_{M_i})$, or the plan avoiding all obstacles P^*, then the search branch is cut, considerably limiting the search tree expansion (Fig.III.8e and III.8f).

[5]The exact operation depends on the world representation being used - more on this in III. 3.2

New Opening detection Computing the third plan step $\mathcal{P}_{to\_goal}$ is by far the most computationally expensive instruction in the neighbor configuration computation loop we previously described. In the worst case (no path/opening to the goal), a full accessible configuration space search needs to be conducted - requiring correspondingly many additional costly robot collision checks[6]. In order to avoid a majority of these worst-case scenarios, a **local** new opening detection search algorithm is used.

The local opening detection algorithm basically relies on the idea that when evaluating the movement of a single obstacle, it can only create a new opening in its own close vicinity. This "close vicinity" is defined as the area occupied by the movable obstacle's footprint inflated by the robot's *diameter* (not radius !). This area is computed for the initial state and final state after the manipulation action applied on obstacle M_i, and respectively written as M_i^{*t} (Fig.III.9a) and M_i^{*t+1} (Fig.III.9b)[7]. The algorithm then computes the Blocking Areas for each state: the geometric intersections between the inflated obstacle and the other obstacles in the environment (Fig.[III.9a, III.9b]). Finally, these Blocking Areas are compared: if at least one of the initial Blocking Areas $\in BA^t$ does not intersect with any of the final Blocking Areas $\in BA^{t+1}$, then it means this initial Blocking Area has been freed, and a local new opening has been created (Fig.III.9c).

This **local** opening detection algorithm guarantees the absence of false negatives - that is, not detecting a local opening when one exists. Otherwise, the full plan would not get computed when it should, making the whole (Wu&Levihn, 2014) algorithm lose its completeness and optimality properties. It may however false positives (detecting a local opening when there is no global opening), which does not affect the overall algorithm's properties, as they will be verified by the full $\mathcal{P}_{to\_goal}$ A^* search.



(A) Obstacle M_3 inflated by robot inscribed circle's diameter at time t (M_3^{*t}) and associated Blocking Areas BA^t (intersections with O_1 & O_2)

(B) Obstacle M_3 inflated by robot inscribed circle's diameter at time $t+1$ (M_3^{*t+1} - blue dots) and associated Blocking Areas BA^{t+1} (intersections with O_1 & outer wall - red dots rectangles)

(C) The three intersection cases between the Blocking Areas: **(i) unblocked area - new local opening detected**, (ii) area still blocked, (iii) newly blocked area

FIGURE III.9: Local Opening Detection example - O_2 no longer blocks the local area around M_3 at $t+1$: new local opening detected

[6]The A Star algorithm has linear time complexity of $\mathcal{O}(|E|)$, E being the number of edges in the graph.

[7]If the robot is not represented by a circle but by a polygon, the circumscribed circle's diameter is used for the initial state inflation, and the inscribed circle's diameter for the final state one - this, as to avoid any false negatives.

III. 2.1.2 Generalization to larger action spaces, continuous environments & edge cases

As we implemented and tested the original algorithm described previously, we realized its limitations and ambiguities in formalization. Thus, we endeavored to completely rewrite the pseudocode formalization, while:

- **generalizing the algorithm** to larger action spaces with arbitrary positive action costs, and to continuous environments representations,
- properly **addressing edge cases** where the original formalization would either fail to find a solution, or enter an infinite loop,
- **unifying notations** in a coherent manner (e.g. multiple assignation operators $\Leftarrow$, $=$ or $\leftarrow$ were unified to $\leftarrow$ to avoid confusions).

Note for the reader The full rewritten algorithms are given here *for reference*. In these algorithms, we will denote the *relevant modifications discussed here* with blue text. Thus, it is not necessary to read the full algorithms to understand the modifications discussed in this section. Although they were completely rewritten because of the original formalization's[8] ambiguities, the original logic textually described in the previous section III. 2.1.1 remains completely relevant, and has been fully preserved. Hence, page numbers to the appropriate textual explanations given in the previous section are provided in the algorithm's headers, should you wish to read them fully.

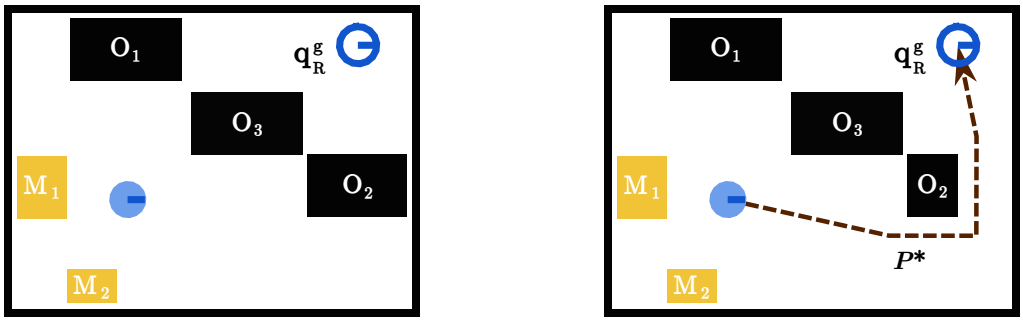
Solving infinite looping edge cases This re-formalization of (Wu&Levihn, 2014)'s algorithm started by properly separating the online plan execution logic from the plan computation logic[9]. We respectively rewrote them as Algorithms 1 (EXECUTE) and 2 (PLAN). This separation actually allowed us to fix a first infinite loop that occurred in the original algorithm, in the edge case where no plan avoiding all obstacles **and** no plan manipulating any of the obstacles were found. This would trivially happen for example, if only static obstacles blocked access to the goal (Fig.III.10a). Now, the once-affected loop (Alg.1, l.5) is properly terminated by returning failure in this case where absolutely no plan could be found (Alg.1, l.10,15).

This same situation where only static obstacles were to block the way to the goal, was actually causing another loop (Alg.2, l.8) to cycle indefinitely. Actually, even if a plan avoiding all obstacles was found, but could not be replaced by a better plan manipulating a movable obstacle (Fig.III.10b), the loop would cycle indefinitely too. This was because the original obstacle routine[10] did not explicitly formalize these cases, which is why we completely rewrote it as Algorithm 3 (SELECT-OBSTACLE). These edge cases now trigger the unambiguous return of a NULL value (Alg.3, l.2-4), that allows the calling loop (Alg.2, l.8) to properly terminate.

[8] Original formalization available in Appendices B and D.

[9] The plan execution and computation logic were originally formalized as a single function called "OPTIMIZED" (Appendix B, Alg.13).

[10] The obstacle choice logic was originally formalized as a function called "GET-NEXT" (Appendix B, Alg.14) + a single-line condition in the calling function "OPTIMIZED" (Appendix B, Alg.13, l.14).



(A) The way to the goal is blocked by static obstacles

(B) A plan avoiding all obstacles exists, but no better plan can be found by manipulating M_1 or M_2

FIGURE III.10: Example edge case scenarios where the original algorithm formalization would enter infinite loops

Algorithm 1: Plan execution routine (EXECUTE) of our improved (Wu&Levihn, 2014) algorithm - Cf. logic description page 57

Data: Robot current configuration q_R^t, Robot goal configuration q_R^g
Result: Success if goal reached, Failure otherwise.

```
1  EXECUTE(q_R^t, q_R^g)
2      mList, eList ← ∅, ∅                    // variables used for obstacle selection in Alg.3
3      P* ← ∅
4      PLAN(P*, q_R^s, q_R^g)
5      while q_R^t ≠ q_R^g do
6          O_new ← O_new ∪ GET-NEW-INFORMATION()   // if failed manipulation → then
            the manipulated obstacle is updated in O_new as static
7          if P* ∩ O_new ≠ ∅ then
8              PLAN(P*, q_R^t, q_R^g)
9              O_new ← ∅
10         if P* ≠ ∅ then
11             nextAction ← GET-NEXT-ACTION(P*)
12             if nextAction ∈ A_M then
13                 mList ← ∅
14             EXECUTE(nextAction)
15         else return Failure
```

For reference, here are the details of the other changes in the Obstacle Choice routine (Alg.3):

- The now recursive nature of the routine serves to properly integrate an originally misplaced condition relative to obstacle choice, that used to be in the calling function "OPTIMIZED" (Appendix B, Alg.13, l.14), and causing an infinite loop when the condition was not verified. It is now expressed as lines 8 & 9 in Algorithm 3, properly ensuring that, when an obstacle M_i is listed in both lists *mList* and *eList*, the entry in *eList* is skipped (as the heuristic cost is less accurate) and return the entry of *mList*, as intended.

- Once an obstacle M_i has been evaluated and thus added or updated in *mList*, we also add it to a new separate list called *evaluated* (Alg.3, l.11-13, initialization in Alg.2, l.3). This

guarantees that the obstacle will not be evaluated twice in the evaluation loop (which could happen in the original formalization).

Algorithm 2: Task-level plan computation routine (PLAN) of our improved (Wu&Levihn, 2014) algorithm - Cf. logic description page 58. Static obstacles are implicitly removed from *eList* and *mList*.

Data: Current optimal plan $\mathcal{P}^*$, Robot current configuration q_R^t, Robot goal configuration q_R^g

```
1  PLAN(P*, q_R^t, q_R^g)
2      eIndex, mIndex ← 0, 0
3      evaluated ← ∅
4      for M_i ∈ O_new do
5          UPDATE(eList, M_i)
6      P* ← A*(W^t, q_R^t, q_R^g)
7      M_i, C_heur ← SELECT-OBSTACLE(eList, mList, eIndex, mIndex, evaluated)
8      while C(P*) ≥ C_heur and M_i ≠ NULL do
9          P, C_l(M_i) = PLAN-FOR-OBSTACLE(M_i, P*, q_R^t, q_R^g)
10         UPDATE(mList, (M_i, C_l(M_i)))
11         if C(P*_{M_i}) < C(P*) then P* ← P*_{M_i}
12         M_i, C_heur ← SELECT-OBSTACLE(eList, mList, eIndex, mIndex, evaluated)
```

Algorithm 3: Obstacle selection routine (SELECT-OBSTACLE) of our improved (Wu&Levihn, 2014) algorithm - Cf. logic description page 59

Data: Heuristic obstacle lists *mList* and *eList*, corresponding indexes *mIndex* and *eIndex*, list of evaluated obstacles *evaluated*

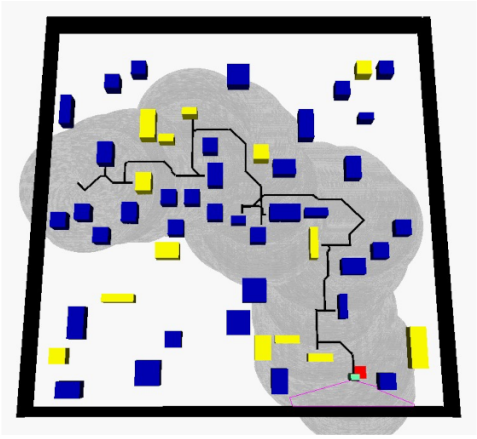
Result: Selected obstacle M_i, corresponding heuristic cost $\mathcal{C}_{heur}$

```
1  SELECT-OBSTACLE(mList, eList, mIndex, eIndex, evaluated)
2      M_i, C_heur ← mList[mIndex]          // M_i, C_heur ← NULL, ∞ if index out of range
3      M_i^e, C_heur^e ← eList[eIndex]      // M_i^e, C_heur^e ← NULL, ∞ if index out of range
4      if C_heur = C_heur^e = ∞ then return NULL, ∞
5      if C_heur ≤ C_heur^e then mIndex++
6      else
7          eIndex++
8          if M_i^e ∈ mList then
9              return SELECT-OBSTACLE(mList, eList, mIndex, eIndex, evaluated)
10         M_i, C_heur ← M_i^e, C_heur^e
11     if M_i ∉ evaluated then
12         APPEND(evaluated, M_i)
13         return M_i, C_heur
14     else return SELECT-OBSTACLE(mList, eList, mIndex, eIndex, evaluated)
```

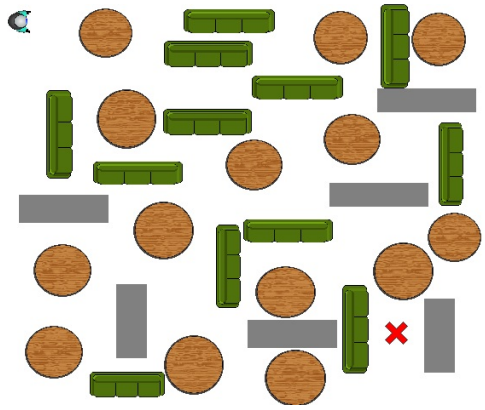
World representation & Robot action space The original algorithm formalization was designed under the following closely coupled hypotheses:

- world representation as a 2D binary occupancy square grid,
- the robot manipulation action space $\mathcal{A}_M$ was restricted to axis-aligned cell-sized translations, with constant action costs $\mathcal{C}(A_M) = C_M \in \mathbb{R}_+^*, \forall A_M \in \mathcal{A}_M$.

While these hypotheses fit the artificial environments used by Wu&Levihn in their experiments (Fig.III.11), they do not fit with challenging (Fig.III.12a) and/or more realistic (Fig.III.12b) environments of our own dataset that is used for experiments in the next chapters (where we discuss the relevance of these scenarios). Fig.III.13 showcases how important rotations are for maneuverability, by "zooming in" on sample situations that may occur in our environments (Fig.III.12). Fig.III.13a illustrates a trivial case where the robot simply cannot find a solution without rotating. On the other hand, Fig.III.13b shows how not allowing rotation would make the algorithm severely less relevant for nonholonomic robots. Indeed, if rotations are forbidden, the only solution is to translate sideways while holding the obstacle. Any robot with nonholonomic drive models, such as differential, Reeds-Shepp or Dubins drives, could only drive in a straight line, thus failing to find a solution.

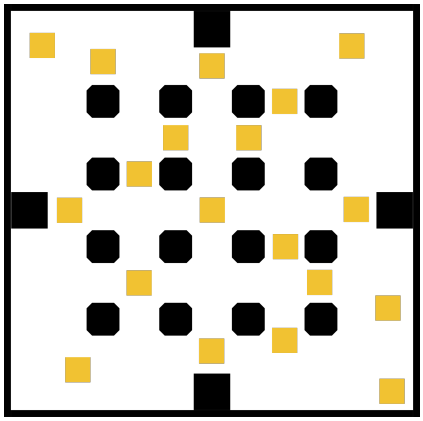


(A) Environment from [33]. Blue obstacles are static, yellow ones are movable, robot is green, goal is red.

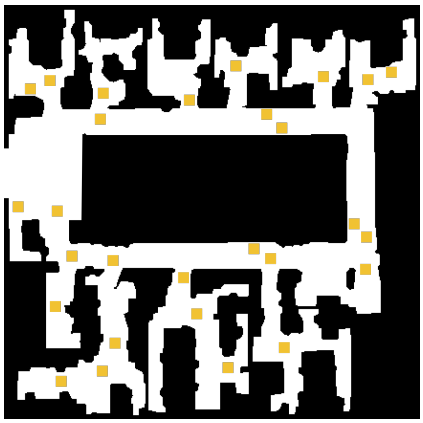


(B) Environment from [7]. Grey obstacles are static, colored ones are movable, robot is blue, goal is red.

FIGURE III.11: Original environments examples used by Wu&Levihn in their experiments. These environments are all artificial, obstacles are placed randomly in axis-aligned directions.



(A) "Intersections"



(B) "CITI Laboratory"

FIGURE III.12: Examples of our experimentation environments in the following chapters: for quite a few obstacles, it is difficult, if not completely impossible for the robot to maneuver while holding them to find relevant manipulation paths.

(A) (Left) Even if the robot is omnidirectional, it cannot pass without rotation. (Right) The robot successfully reaches the goal by rotating the obstacle counter-clockwise.

(B) (Left) A nonholonomic robot cannot solve this problem without rotation. (Right) An omnidirectional robot may however solve it by translating sideways with the obstacle.

FIGURE III.13: Zoomed-in sample situations that may occur in Fig.III.12, where the relevance of rotation is shown.

As mentioned in the previous section III. 2.1.1 (page 59), the original obstacle manipulation planning routine used a customized breadth-first search algorithm[11], but over a limited set of four axis-aligned, cell-sized translations. **We generalized this search, by first rewriting it as a standard Dijkstra algorithm call** (Alg.4, l.7). We parameterized the search with a method that computes the valid neighbor robot configurations from the current one by exploring a generic manipulation action space $\mathcal{A}_{\mathcal{M}}$ (Alg.4, l.11-12). The only constraint for this action space $\mathcal{A}_M$, is to yield a finite configuration space - that is, a finite graph[12]. All the customized logic from the original breadth-first search has been extracted and encapsulated in this function[13]. Using Dijkstra's algorithm allows actions to have arbitrary positive costs, which are more relevant than constant costs for a generic action space. As a side note, this rewrite also allowed us to properly formalize the iteration over grasping configurations (Alg.4, l.4).

We also wrote the continuous geometry equivalent (Alg.5) to the efficient local opening detection algorithm[14] originally written for a discrete regular 2D binary occupancy grid. By doing so, **we have generalized the original algorithm to continuous geometry spaces**, where, among other possibilities, computing rotations can be done without geometrical information loss. In other words, rasterizing obstacles is no longer a necessity for the algorithm to be used.

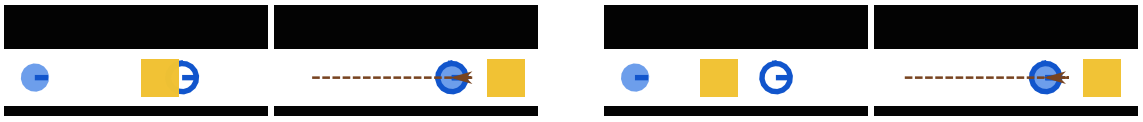
Solving Obstacle-on-Goal problems In [34], Levihn emitted the hypothesis that the algorithm only works under the assumption that the goal configuration is completely "free of obstacles". However, this was not further discussed leaving it to the reader to understand why the algorithm could not deal with situations as shown in Fig.III.14 - cases that very often occurred in our multi-Robot experiments in Chapter V. Actually, we discovered it all came down to the original efficient opening detection algorithm[14] that would not detect a new local opening in these cases, resulting in the algorithm not finding a plan. Thus, as we rewrote it (Alg.5), we solved this by simply adding the goal configuration geometry as an obstacle in the function call (Alg.4, l.25, noted as $W^t_{\cup R^g / M_i, R}$).

[11] This customized breadth-first search algorithm was originally formalized as a function called "OPT-EVALUATE-ACTION" (Appendix B, Alg.15).

[12] This is a precondition for Dijkstra's algorithm to be complete and optimal (and thus, the overall algorithm to remain optimal too).

[13] The main difference between breadth-first search and Dijkstra's algorithm is the use of a more computationally demanding priority queue instead of a queue. Our implementation automatically detects if action costs are constant in which case, the queue types are transparently switched - making it computationally equivalent.

[14] Formalized in [35], formalization copied in Appendix D.



(A) (Left) The original algorithm can not find a solution, as the goal is covered by an obstacle. (Right) Our modified algorithm successfully clears the goal by translating the obstacle further.

(B) (Left) The original algorithm can not find a solution, as the goal would be covered during manipulation. (Right) Our modified algorithm successfully clears the goal by translating the obstacle further.

FIGURE III.14: Zoomed-in sample situations that may occur in Fig.III.12, where the robot needs to move an obstacle over the goal.

Algorithm 4: Obstacle manipulation planning routine (PLAN-FOR-OBSTACLE) of our improved (Wu&Levihn, 2014) algorithm - Cf. logic description page 59

Data: Obstacle to evaluate M_i, current optimal plan P^*, current robot configuration q_R^t, goal robot configuration q_R^g

Result: Optimal manipulation plan for the evaluated obstacle $\mathcal{P}^*_{M_i}$, corresponding heuristic cost $\mathcal{C}_l(M_i)$

PLAN-FOR-OBSTACLE(M_i, P^*, q_R^t, q_R^g)
  $\mathcal{P}^*_{M_i} \leftarrow \emptyset$
  $\mathcal{C}_l(M_i) \leftarrow \infty$
  for $q_R^{manip} \in$ VALID-GRASPS(M_i) **do**
    $\mathcal{P}_{to\_obstacle} \leftarrow A^*(W^t, q_R^t, q_R^{manip})$
    if $\mathcal{P}_{to\_obstacle} = \emptyset$ **then skip**
    DIJKSTRA(q_R^{manip}, q_R^g, GET-NEIGHBORS-MANIP)
  return $\mathcal{P}^*_{M_i}$, $\mathcal{C}_l(M_i)$
GET-NEIGHBORS-MANIP(q_R^{cur})
  $neighbors \leftarrow \emptyset$
  for $A_M \in \mathcal{A}_M$ **do**
    $q_R^{next}, q_{M_i}^{next} \leftarrow$ APPLY-ACTION($(q_R^{cur}, q_{M_i}^{cur})$, A_M)
    if $q_R^{next} \in closedList$ **then skip**
    $\mathcal{C}(\mathcal{P}_{manip}) \leftarrow gScore[q_R^{cur}] + \mathcal{C}(A_M)$
    $\mathcal{C}_{est} \leftarrow \mathcal{C}(\mathcal{P}_{to\_obstacle}) + \mathcal{C}(\mathcal{P}_{manip}) + |q_R^g - q_R^{next}|$
    if $\mathcal{C}_{est} \geq \mathcal{C}(\mathcal{P}^*_{M_i})$ **or** $\mathcal{C}_{est} \geq \mathcal{C}(\mathcal{P}^*)$ **then skip**
    if $q_R^{next} \cap W^t_{/M^i,R} \neq \emptyset$ **or** $q_{M_i}^{next} \cap W^t_{/M^i,R} \neq \emptyset$ **then skip**
    APPEND($neighbors$, q_R^{next})
    if CHECK-NEW-OPENING($W^t_{\cup R^g/M_i,R}$, M_i^{cur}, M_i^{next}) **then**
      $\mathcal{P}_{to\_goal} \leftarrow A^*(W^{next}, q_R^{next}, q_R^g)$
      if $\mathcal{P}_{to\_goal} \neq \emptyset$ **then**
        if $\mathcal{C}(\mathcal{P}_{manip}) + \mathcal{C}(\mathcal{P}_{to\_goal}) < \mathcal{C}_l(M_i)$ **then**
          $\mathcal{C}_l(M_i) \leftarrow \mathcal{C}(\mathcal{P}_{manip}) + \mathcal{C}(\mathcal{P}_{to\_goal})$
        if $\mathcal{C}(\mathcal{P}_{to\_obstacle}) + \mathcal{C}(\mathcal{P}_{manip}) + \mathcal{C}(\mathcal{P}_{to\_goal}) < \mathcal{C}(\mathcal{P}^*_{M_i})$ **then**
          $\mathcal{P}_{manip} \leftarrow$ RECONSTRUCT-PATH($cameFrom$, q_R^{manip}, q_R^{cur})
          $\mathcal{P} \leftarrow \mathcal{P}_{to\_obstacle} + \mathcal{P}_{manip} + \mathcal{P}_{to\_goal}$
          $\mathcal{P}^*_{M_i} \leftarrow \mathcal{P}$
  return $neighbors$

Algorithm 5: New opening detection routine (CHECK-NEW-OPENING) for continuous geometries - Cf. logic description page 62

Data: Current world state without the polygons of the robot and the obstacle to evaluate, but with the polygon of the robot if it were on the goal $W^t_{\cup R^g/M_i,R}$, Current state of the obstacle to evaluate M^t_i, next state of the obstacle to evaluate after action M^{t+1}_i

Result: True if new local opening detected, False otherwise

```
1  CHECK-NEW-OPENING(W^t_{∪R^g/M_i,R}, M^t_i, M^{t+1}_i)
2      M^{*t}_i ← INFLATE-BY-ROBOT-CIRCUMSCRIBED-CIRCLE-DIAMETER(M^t_i)
3      M^{*t+1}_i ← INFLATE-BY-ROBOT-INSCRIBED-CIRCLE-DIAMETER(M^{t+1}_i)
4      BA^t ← W^t_{/M_i,R} ∩ M^{*t}_i
5      BA^{t+1} ← W^t_{/M_i,R} ∩ M^{*t+1}_i
6      for BA^t_i ∈ BA^t do
7          if not IS-STILL-BLOCKED(BA^t_i, BA^{t+1}) then return True
8      return False
9  IS-STILL-BLOCKED(BA^t_i, BA^{t+1})
10     for BA^{t+1}_j ∈ BA^{t+1} do
11         if BA^t_i ∩ BA^{t+1}_j then return True
12     return False
```

As a concluding remark, we underline our improvements constitute a generalization, and not just a different interpretation of the algorithm. This being because our formalization is also capable of planning under the same original constraints with equivalent computational requirements.

III. 2.1.3 When more knowledge becomes an issue

As we have seen in the previous sections, (Wu&Levihn, 2014)'s algorithm *can only generate plans manipulating at most a single obstacle* (Cf. page 58). For instance, in the scenario illustrated in Fig.III.15a, the algorithm's task-level planner can not compute a plan that would contain the transfer paths necessary to manipulate the two movable obstacles. That is why the algorithm was designed by Wu&Levihn under the hypotheses that, as illustrated in Fig.III.15b:

- *the robot can only acquire knowledge through its own limited sensing capabilities*,
- *all space not yet observed is free space*,
- and *obstacles are to be considered movable until manipulating them fails*.

This way, the algorithm does not have to globally consider a NAMO problem, but can locally solve smaller one-obstacle problems, as it discovers them (Fig.III.15c).

This approach however makes the algorithm's resolution capabilities extremely dependent on the robot's sensing capabilities, such as the field of view's size and shape. For instance, in the previous example, if the field of view's radius (Fig.III.15b) were big enough to cover both movable obstacles, the algorithm would not be able to solve the problem. With such an approach, *more knowledge or better sensing capabilities do not necessarily lead to better results*, and may even prevent the algorithm from finding a solution.

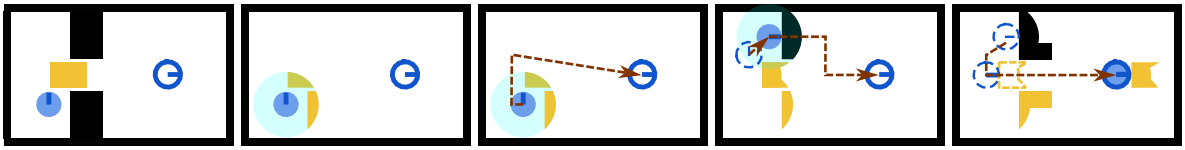
(A) Problem full view - The algorithm can't solve it as is, as it requires two transfers in a plan.

(B) Without any prior knowledge, the robot only perceives obstacles within its field of view

(C) The problem can thus be solved as new plans are computed as obstacles are discovered.

FIGURE III.15: (Wu&Levihn, 2014)'s algorithm : Unsolvable problem using full prior knowledge (A), but solvable using a limited (clear blue) circular field of view (B, C).

Upon reading this first example (Fig.III.15), one may be tempted to think that if we only provided prior information on static obstacles, the algorithm would still find a solution ? The answer is no, as shown in the counter-example of Fig.III.16.



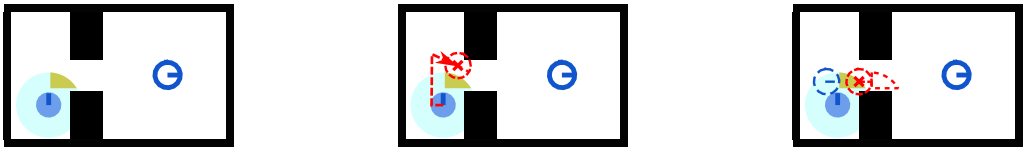
(A) Initial world state (ground truth)

(B) Initial world perception - **No prior world knowledge provided**

(C) First plan circumvents the perceived movable obstacles

(D) Plan is updated as obstacles are discovered, until failure to move the top one.

(E) Plan is updated to move the middle obstacle and goal is finally reached.



(F) Initial world perception - But this time, **knowledge about static obstacles is provided beforehand**

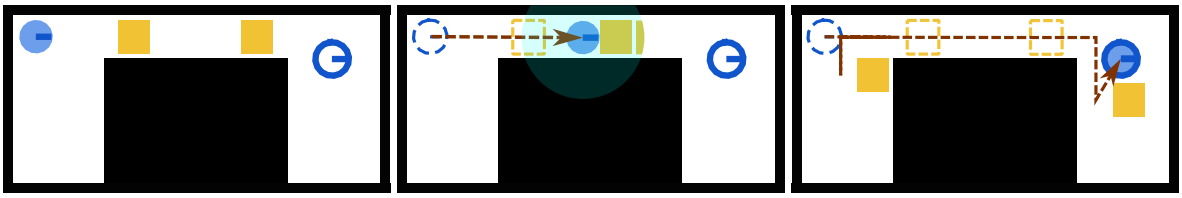
(G) The robot tries to plan while avoiding the obstacles, but fails to find a non-colliding plan.

(H) The robot tries to plan a plan by moving the middle obstacle, but fails as the robot's body would collide with the bottom obstacle.

FIGURE III.16: Example showcasing how even partial prior environment knowledge may negatively affect the ability of (Wu&Levihn, 2014)'s algorithm - [B, C, D, E] assume no prior knowledge, while [F, G, H] assume static obstacles are prior knowledge. The example assumes that the robot manipulation action space is limited to forward translations (pushes).

The conclusion of this observation however is not that using prior knowledge is completely irrelevant. Rather, it is that if planning first using prior knowledge fails (Fig.III.16 [F, G, H]), then planning a second time only with sensed data may still succeed (Fig.III.16 [B, C, D, E])[15]. But still, these limitations can very easily lead to bad planning choices that end up blocking the robot, as illustrated in Fig.III.17. This thus calls for an alternative algorithm that can produce plans with multiple transfer paths, without having to forego prior knowledge when it is available: this is what we shall discuss in the next section with (Stilman, 2005)'s algorithm.

[15]The provided pseudocode can be trivially modified for this by calling the PLAN function (after Alg.1 line 7) a second time if $\mathcal{P}^* = \emptyset$, but this time stripping the world representation W^t of any prior knowledge / only keeping the accumulated sensed data.



(A) Initial world state (ground truth) - As in Fig.III.15a, (Wu&Levihn, 2014)'s algorithm cannot find a solution with full prior knowledge.

(B) Without prior knowledge, (Wu&Levihn, 2014)'s algorithm ends up blocking itself.

(C) (Stilman, 2005)'s solves the problem with full prior knowledge (Cf. following section).

FIGURE III.17: Unsolvable case for (Wu&Levihn, 2014)'s algorithm - whatever its sensing capabilities, and solution provided by (Stilman, 2005)'s algorithm.

III. 2.2 (Stilman, 2005)'s Algorithm

As presented in Chapter II, Stilman proposed the first - and still the only - resolution complete algorithm for a class of problems called LP1 [3]. As discussed in Chapter II, an LP1 problem is a NAMO problem that can be decomposed in 1-obstacle NAMO sub-problems (keyholes), where the solution to each sub-problem does not interfere with the solution of the other sub-problems. This means that a NAMO plan solving a Linear Problem of type 1 (LP1) may contain several obstacle manipulations.

In these terms, (Wu&Levihn, 2014)'s algorithm guarantees completeness and optimality for a subclass of LP1 problems, where the sequence of sub-problems is actually a single problem (one obstacle manipulation). Comparatively, we could say that (Stilman, 2005)'s algorithm thus (partially) drops optimality in exchange for completeness for the entire class of LP1 problems - the reason why we chose to use it as a base in the next chapters.

III. 2.2.1 Algorithm outline

In this section, we present the original structure of (Stilman, 2005)'s algorithm, as we did with (Wu&Levihn, 2014)'s algorithm. Again, our improvements will be subsequently highlighted in the next section.

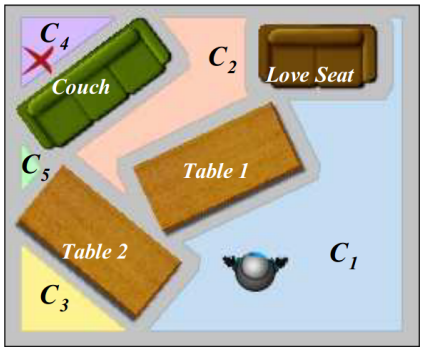
Task-level planner (Stilman, 2005)'s algorithm was designed as an offline planning algorithm. Hence, it does not provide a plan execution routine, but only a task-level planner and its obstacle selection and motion planning subroutines. Thus, (Stilman, 2005)'s algorithm can be used as is only under the assumption of omniscient knowledge about the environment (obstacles' geometry, type and position are known with no uncertainty)[16]. The task-level planner either yields:

- a transit path to the goal q_R^g if one exists (by means of an A^* star search over the navigation action space),

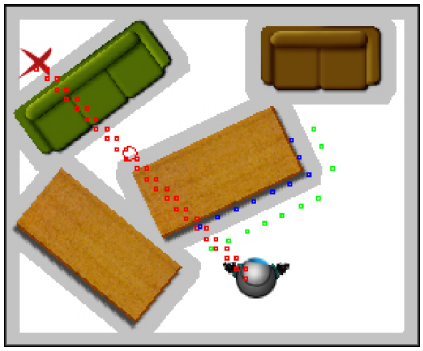
[16]It is however possible to use (Stilman, 2005)'s algorithm under the same perception hypotheses as (Wu&Levihn, 2014)'s algorithm. This can be achieved by using the same improved execution routine (Alg. 1) we previously formalized.

- **or** a sequence of alternating transit and transfer paths until a transit path to the goal q_R^g can be computed (illustrated by Fig.III.18).

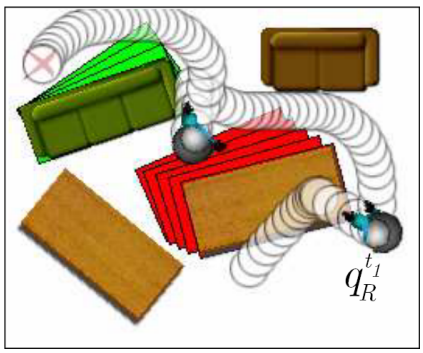
As such, in contrast with (Wu&Levihn, 2014)'s algorithm, (Stilman, 2005)'s algorithm will not even try to find a plan moving obstacles if a direct transit path to the goal exists. This is the main reason why it can not be proven to be displacement-cost optimal.



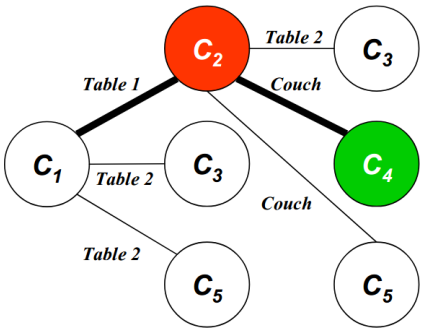
(A) Initial world state with color-filled Components $C_R^{free} = \{C_1, C_2, C_3, C_4, C_5\}$



(B) Red dots are a path traversing obstacles, produced by the heuristic obstacle selection planner. Blue dots are grasping points, green dots are corresponding robot manipulation configurations.



(C) Solution yielded by (Stilman, 2005)'s algorithm



(D) Task-level search tree, with obstacles as edges and components as nodes

FIGURE III.18: LP1 problem example copied from [19]. The robot navigates in C_1 to transfer the *Table 1* to fuse C_1 with C_2, then navigates to the Couch to fuse C_2 with C_4 in order to finally navigate to its goal.

In order to decompose LP1 NAMO problems into independent 1-obstacle sub-problems, the algorithm relies on the underlying structure of space, expressed as Components of free space, shown as colored areas in Fig.III.18a. The general idea of the algorithm is thus to try and fuse the component where the robot originally is (C_1 in fig.), to the one where its goal configuration q_R^g is (C_4 in fig.) and every relevant intermediate component in between (C_2 in fig.), by each time moving a single obstacle (here, the *Table 1* then the *Couch*). Since the algorithm was proven complete in the resolution of LP1 problems [3, 19], failure to find a plan would mean the problem was of a harder class than LP1 - or that no solution exists.

Structurally, the task-level planner is formulated as a recursive forward best-first search of the best tuple (obstacle to move M_F, component to fuse with C_F) - e.g. (M_F = *Table 1*, $C_F = C_2$) in Fig.III.18. Once the pair is selected by using the obstacle selection routine (described a bit

further below), a transfer path P_{manip} is computed, by searching over the robot's manipulation action space $\mathcal{A}_M$ until an opening is created - similarly to (Wu&Levihn, 2014)'s algorithm. If so, then the obstacle selection and manipulation planning sequence is recursively executed until a transit path $\mathcal{P}_{to\_goal}$ to the goal q_R^g is reached or deemed unreachable. As the recursive loop backtracks after reaching the goal, the intermediate transit path to the selected obstacle $P_{to\_obstacle}$ is computed, and both paths are added to the overall plan. This process results in the final sequence of alternating transit/transfer paths leading to the goal: the NAMO plan.

Obstacle selection Stilman's obstacle selection routine inspired many of the subsequent works (in particular, but not only [46, 4, 51]) and is a core element of the algorithm. This routine selects the next obstacle to move/component to fuse with (M_F, C_F) pair, by using a relaxed-constraint path planner (A^* variant) that is allowed to plan *through obstacles*. This planner returns the *first* obstacle M_F and component C_F it encountered in the search branch that reached the goal first, following the 4 allowed transitions below, illustrated in Fig.III.19:

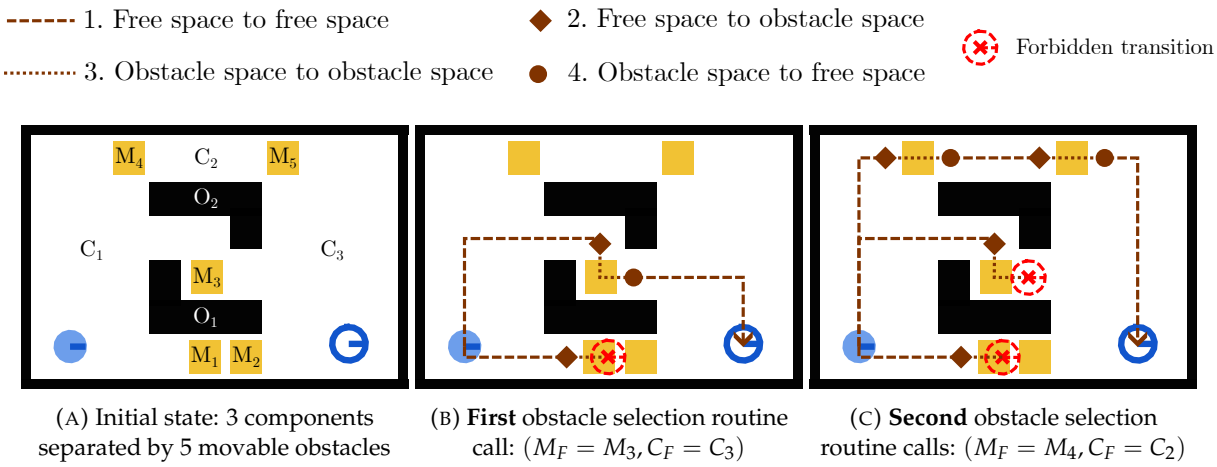
1. **Free space to free space:** from a collision-free robot configuration to another (formally: $q_R^{cur}, q_R^{next} \in C_R^{free}$);
2. **Free space to obstacle space:** from a collision-free robot configuration to a robot configuration that intersects ***exclusively*** *with a single movable obstacle* M_i (formally: $q_R^{cur} \in C_R^{free}$ **and** $q_R^{next} \in$ **exc** $\chi_R^{M_i}$);
3. **Obstacle space to obstacle space:** from a robot configuration that intersects *with a single movable obstacle* M_i to a robot configuration that intersects *with the same single movable obstacle* M_i (formally: $q_R^{cur}, q_R^{next} \in$ **exc** $\chi_R^{M_i}$);
4. **Obstacle space to free space:** from a robot configuration that intersects *with a single movable obstacle* M_i to a collision-free robot configuration (formally: $q_R^{cur} \in$ **exc** $\chi_R^{M_i}$ **and** $q_R^{next} \in C_R^{free}$).

In Fig.III.19b, we can see how the search tree is cut short as it tries to traverse M_2 after M_1, as it would violate transition 3 (obstacle space to obstacle space). That is why the tree is extended to pass through M_3, eventually reaching the goal in component C_3. However, there is not enough room around M_3 to manipulate it successfully to connect C_3 with C_1: the manipulation planner fails to move M_3 resulting in a **second** call of the obstacle selection routine in Fig.III.19c. In order for this second call and any subsequent ones to not indefinitely return the same obstacle/component pair (M_F, C_F), a list of these pairs is kept track of in the task-level planner (see[17] for a detailed explanation).

[17] The obstacle selection routine then uses this list to forbid transitions from an obstacle to free space (4.) when the following conditions are met:

- the traversed obstacle M_i is the first one to be traversed (formally: $M_F = M_i \neq 0$),
- **and** no other component than the initial one has been traversed yet (formally: $C_F = 0$),
- **and** the neighbor robot configuration belongs to a component C_i so that the pair (M_i, C_i) is not in the list of evaluated pairs,
- **and** C_i has not previously been fused by the task planner.

This is why during the second call to the obstacle selection routine, the search tree is cut short after traversing M_3, resulting in the tree being grown through M_4 and M_5, eventually leading to a solution to the problem showcased in III.19. If we go back to Fig.III.18b, an actual path produced by the obstacle selection routine can be seen. This path first traverses $M_F =$ *Table 1* to enter $C_F = C_2$. The selection routine is called again, starting from intermediate robot configuration $q_R^{t_1}$, returning $M_F =$ *Couch* to enter $C_F = C_4$ (not illustrated). This is how the full plan shown in Fig.III.18c and III.18d is found.



(A) Initial state: 3 components separated by 5 movable obstacles

(B) **First** obstacle selection routine call: $(M_F = M_3, C_F = C_3)$

(C) **Second** obstacle selection routine calls: $(M_F = M_4, C_F = C_2)$

FIGURE III.19: Example NAMO scenario showcasing the obstacle selection routine. The four types of allowed transitions are shown as brown lines and markers. For the sake of simplicity, only axis-aligned translations are allowed and only the relevant branches of the A^* search are shown instead of the full tree.

Obstacle manipulation planning Once the obstacle selection subroutine returns the obstacle / component pair (M_F, C_F) to evaluate to the task-level planner, it is passed along to the obstacle manipulation planning routine. This routine is a breadth-first search over the robot's manipulation action space $\mathcal{A}_M$, starting from the valid manipulation configurations q_R^{manip} where the robot can grasp the obstacle (Fig.III.18b). The search is stopped as soon as an opening is created - that is, when an A^* call over the robot's navigation action space $\mathcal{A}_\mathcal{N}$ returns a valid path $\mathcal{P}_{to\_C_F}$, from the currently evaluated configuration q_R^{cur}, to an arbitrary configuration $q_R^{C_F}$ in the component to fuse with (C_F). The search prioritizes configurations in the free space component the robot comes from C_R^{acc}, as to avoid interfering with a future obstacle manipulation in the target component C_F (hence why the robot pulls the obstacles back in Fig III.18c).

III. 2.2.2 Generalization to larger action spaces & edge cases

Like we did for (Wu&Levihn, 2014)'s algorithm, we similarly completely rewrote the pseudocode formalization of (Stilman, 2005)'s algorithm in a more generic fashion. This was done with the same goals of **generalizing the algorithm** to larger action spaces with arbitrary positive action costs, properly **addressing the obstacle-on-goal edge case** where the original formalization would fail to find a solution, and **unifying notations** in a coherent manner with

those of (Wu&Levihn, 2014)'s algorithm. Additionally, we also **improved overall computational performance**, by applying the local new opening detection algorithm previously described (Section III. 2.2.2, Alg.5).

Note for the reader Again, as for (Wu&Levihn, 2014)'s algorithm, The full rewritten algorithms are given here *for reference*. In these algorithms, we will denote the *relevant modifications discussed here* with blue text. Thus, it is not necessary to read the full algorithms to understand the modifications discussed in this section. Although they were completely rewritten because of the original formalization's[18] ambiguities, the original logic textually described in the previous section III. 2.2.1 remains completely relevant, and has been fully preserved. Hence, page numbers to the appropriate textual explanations given in the previous section are provided in the algorithm's headers, should you wish to read them fully.

Computing the free space components How the nodes (free space components) of the task-level search tree illustrated in Fig.III.18d[19] are determined is actually not addressed by Stilman. However, as presented in the previous section III. 2.2.1, knowing which free space component C_i contains a given robot configuration is essential, as this is how obstacles are selected for manipulation evaluation. In the absence of specification, we can only suppose that free space components are initialized before the task-level planning routine is executed. We consequently wrote Algorithm 6, which first executes a recursive Breadth-First Search over the robot navigation action space $\mathcal{A}_N$, that creates a search tree for each free space component C_i and each space component where the robot would intersect with a single movable obstacle $\chi_R^{M_j}$[20]. Only then is the task-level planner (Alg.7) executed.

Solving Obstacle-on-Goal problems Just like the original (Wu&Levihn, 2014) algorithm, (Stilman, 2005)'s algorithm originally only works under the assumption that the goal configuration is completely free of obstacles. While this choice is perfectly justified, as it facilitated their demonstration of completeness for LP1 problems (requiring, by definition, the goal robot configuration to belong to a free space component), it prevented the algorithm from solving the same simplistic examples as described in Fig.III.14a.

This obstacle-on-goal problem was solved by rewriting the obstacle selection routine as Algorithm 8. It is now formulated as a generic A^* call (l.3), moving the extra logic of the original customized A^*[21] in a separate method, as we did for (Wu&Levihn, 2014)'s algorithm (Alg.4). In this extra logic, we removed a condition that returned a NIL value in the original formalization for both the following cases:

[18] Original formalization available in Appendix C.

[19] This search tree is named Free Components Tree abb. FCT by Stilman in his thesis [19].

[20] As each search tree only covers its associated free space component or movable obstacle, the overall computational cost is equivalent to that of a regular Breadth-First Search over the robot navigation action space $\mathcal{A}_N$, under the same assumption of being allowed to traverse a single obstacle at once. In other words, each tree is basically just a branch of the regular Breadth-First Search.

[21] The original obstacle selection routine is available in Appendix B, Alg.17.

Algorithm 6: Top-level plan computation routine (PLAN) of our improved (Stilman, 2005) algorithm

Data: Current world state W^t, goal configuration q_R^g
Result: A valid NAMO Plan P or empty plan $\emptyset$

PLAN(W^t, q_R^g)
  if $R^g \cap W^t_{/R,M} \neq \emptyset$ **or** ($q_R^g \notin C_R^{free}$ **and** $q_R^g \notin$ **exc** $\chi_R^{M_j}$) **then return** $\emptyset$
  $i, C_R^{t_i} \leftarrow 0, \{C_i : \text{NULL}\}$
  $C_R^{t_0}[C_i] \leftarrow$ GET-CLOSED-SET(BREADTH-FS(W^t, GET-NEIGHBORS-CC))}
  return SELECT-CONNECT(W^t, $[C_0]$, q_R^g, $C_R^{t_0}$)
GET-NEIGHBORS-CC(q_R^{cur})
  neighbors $\leftarrow \emptyset$
  for $A_N \in \mathcal{A}_N$ **do**
    $q_R^{next} \leftarrow$ APPLY-ACTION(q_R^{cur}, A_N)
    if $q_R^{next} \in$ *closedList* **or** $q_R^{next} \cap W^t_{/R,M}$ **then skip**
    if $q_R^{cur}, q_R^{next} \in C_R^{free}$ **or** $q_R^{cur}, q_R^{next} \in$ **exc** $\chi_R^{M_j}$ **then** APPEND(*neighbors*, q_R^{next})
    else if $q_R^{cur} \in C_R^{free}$ **and** $q_R^{next} \in$ **exc** $\chi_R^{M_j}$ **and** $\chi_R^{M_j} \notin C_R^{t_0}$ **then**
      $C_R^{t_0}[\chi_R^{M_j}] \leftarrow$ NULL
      $C_R^{t_0}[\chi_R^{M_j}] \leftarrow$ GET-CLOSED-SET(BREADTH-FS(W^t, GET-NEIGHBORS-CC))}
    else if $q_R^{cur} \in$ **exc** $\chi_R^{M_j}$ **and** $q_R^{next} \in C_R^{free}$ **and** $C_i \notin C_R^{t_0}$ **then**
      $i, C_R^{t_0}[C_i] \leftarrow i+1$, NULL
      $C_R^{t_0}[C_i] \leftarrow$ GET-CLOSED-SET(BREADTH-FS(W^t, GET-NEIGHBORS-CC))}
  return *neighbors*

Algorithm 7: Task-level plan computation routine (SELECT-CONNECT) of our improved (Stilman, 2005) algorithm - Cf. logic description page 71

Data: Current world state W^t, components fused in previous recursions *FusedComponents*, goal configuration q_R^g, initial connected components $C_R^{t_0}$
Result: A valid NAMO Plan P or empty plan $\emptyset$

SELECT-CONNECT(W^t, *FusedComponents*, q_R^g, $C_R^{t_0}$)
  EvaluatedPairs $\leftarrow \emptyset$
  if $\mathcal{P}_{to\_goal} \leftarrow A^*(W^t, q_R^t, q_R^g) \neq \emptyset$ **then return** ($\mathcal{P}_{to\_goal}$)
  while $M_F, C_F \leftarrow RCH(W^t, EvaluatedPairs, FusedComponents, q_R^g, C_R^{t_0}) \neq (0,0)$ **do**
    $W^{t+2}, \mathcal{P}_{manip} \leftarrow$ MANIP-SEARCH(W^t, M_F, C_F, $C_R^{t_0}$)
    if $\mathcal{P}_{manip} \neq \emptyset$ **then**
      FuturePlan $\leftarrow$ SELECT-CONNECT(W^{t+2}, *FusedComponents* + $[C_F]$, q_R^g, $C_R^{t_0}$)
      if *FuturePlan* $\neq \emptyset$ **then**
        $\mathcal{P}_{to\_obstacle} \leftarrow A^*(W^t, q_R^t, \mathcal{P}_{manip}[0])$
        return ($\mathcal{P}_{to\_obstacle} + \mathcal{P}_{manip}$ + *FuturePlan*)
    EvaluatedPairs **append** (M_F, C_F)
  return $\emptyset$

- when **a single obstacle** M_F intersects with the goal configuration q_R^g (as in Fig.III.14a), now the function returns a $(M_F, 0)$ tuple,
- when **more than one** movable obstacle intersects with the goal configuration q_R^g, or when the **goal is unreachable**, now the function returns a (0, 0) tuple, so that planning failure only occurs in this case - (Alg.7, l.4)[22].

Algorithm 8: Obstacle selection routine (RCH) of our improved (Stilman, 2005) algorithm - Cf. logic description page 73

Data: World state W^t, previously evaluated (M_i, C_i) pairs *EvaluatedPairs*, components fused in previous recursions *FusedComponents*, goal configuration q_R^g, initial connected components $C_R^{t_0}$

Result: Relevant obstacle to consider for manipulation O_F, Connected Component C_F to be joined with currently accessible cells C_R^{acc}

1 `RCH`(W^t, *EvaluatedPairs*, *FusedComponents*, q_R^g, $C_R^{t_0}$)
2 $MC \leftarrow \{q_R^t : (0,0)\}$
3 $q_R^{cur} \leftarrow A^*(q_R^t, q_R^g$, GET-NEIGHBORS-RCH)
4 **if** $q_R^{cur} = q_R^g$ **then return** $MC[q_R^{cur}]$
5 **else return** (0, 0)
6 `GET-NEIGHBORS-RCH`(q_R^{cur})
7 $neighbors \leftarrow \emptyset$
8 $M_F, C_F \leftarrow MC[q_R^{cur}]$
9 **for** $A_N \in \mathcal{A}_N$ **do**
10 $q_R^{next} \leftarrow$ APPLY-ACTION((q_R^{cur}), A_N)
11 **if** $q_R^{next} \in closedList$ **then skip**
12 **if** $S(q_R^{cur}, q_R^{next}) \cap W^t_{/R,M}$ **or** $(q_R^{next} \notin C_R^{free}$ **and** $q_R^{next} \notin$ **exc** $\chi_R^{M_i})$ **then skip**
13 **if** $(C_F \neq 0)$ **then** $MC[q_R^{next}] \leftarrow (M_F, C_F)$
14 **else if** $(M_F \neq 0$ **and** q_R^{next} **exc** $\chi_R^{M_F})$ **then** $MC[q_R^{next}] \leftarrow (M_F, 0)$
15 **else if** $(M_F \neq 0$ **and** $q_R^{next} \in C_i$ **s.t.** $C_i \notin FusedComponents$ **and** $(M_F, C_i) \notin EvaluatedPairs)$ **then**
16 $MC[q_R^{next}] \leftarrow (M_F, C_i)$
17 **else if** $(M_F = 0$ **and** $q_R^{next} \in C_R^{free})$ **then** $MC[q_R^{next}] \leftarrow (0,0)$
18 **else if** $(M_F = 0$ **and** $q_R^{next} \in$ **exc** $\chi_R^{M_i})$ **then** $MC[q_R^{next}] \leftarrow (M_i, 0)$
19 **if** $q_R^{next} \in MC$ **then** APPEND($neighbors$, q_R^{next})
20 **return** *neighbors*

Formalizing MANIP-SEARCH The main source of ambiguity of (Stilman, 2005)'s original work was the absence of a pseudocode formalization of the obstacle manipulation planning routine "MANIP-SEARCH". From the various textual observations, we formalized it as true to the original descriptions as possible, in Appendix C (Alg.18). The version provided in this section has been improved similarly to (Wu&Levihn, 2014)'s algorithm to use a Dijkstra search

[22] The manipulation search procedure has correspondingly been adapted (Alg.9, l.19-20) to manage this goal-over-obstacle case, by having the exit condition be the reachability of the robot goal configuration q_R^g instead of a random robot configuration in C_F, since in this case $C_F = 0$

(Alg.9, l.2) instead of a Breadth-First Search[23], allowing for actions to have any positive cost instead of a constant cost. Also, we incorporated the local new opening detection algorithm from (Wu&Levihn, 2014)'s algorithm (Cf. page 62). Now the A^* search to verify if a global opening has been created is only executed if a new local opening is found (Alg.9, l.18).

Algorithm 9: Obstacle manipulation planning routine (MANIP-SEARCH) of our improved (Stilman, 2005) algorithm - Cf. logic description page 74

Data: Current world state W^t, Relevant obstacle to consider for manipulation O_F, Connected Component C_F to be joined with currently accessible cells C_R^{acc}, initial connected components $C_R^{t_0}$

Result: Evaluated world state after manipulation W^{t+2}, Manipulation plan τ_M

```
1  MANIP-SEARCH(W^t, O_F, C_F, C_R^{t_0})
2      τ_M ← DIJKSTRA((∅,∅), GET-NEIGHBORS, EXIT-CONDITION)
3      W^{t+2} ← COPY-AND-UPDATE(W^t, τ_M)
4      return W^{t+2}, τ_M
5  GET-NEIGHBORS((q_R^{manip}, q_R^{cur}))
6      neighbors ← ∅
7      if (q_R^{manip}, q_R^{cur}) = (∅,∅) then
8          for q_R^{manip} ∈ VALID-GRASPS(M_i) do
9              APPEND(neighbors, (q_R^{manip}, q_R^{manip}))
10         return neighbors
11     for A_M ∈ 𝒜_M do
12         R^{next}, M_i^{next} ← APPLY-ACTION((R^{cur}, M_i^{cur}), A_M)
13         if q_R^{next} ∈ closedList then skip
14         if S(A_M) ∩ W^t_{/M^i,R} ≠ ∅ then skip
15         APPEND(neighbors, q_R^{next})
16     return neighbors
17 EXIT-CONDITION(q_R^{cur})
18     if CHECK-NEW-OPENING(W^t_{∪R^g / M_i,R}, M_i, A_M) then
19         q_R^{C_F} ← q_R^g
20         if q_R^g ∉ C_F and C_F ≠ 0 then q_R^{C_F} ← RANDOM-FREE-CONFIGURATION(W_t, C_F)
21         P_{to_C_F} ← A*(q_R^{cur}, q_R^{C_F})
22         if P_{to_C_F} ≠ ∅ then return True
23     return False
```

In this section, we have presented both of our revised algorithms. Let us now present the new simulation tool we have written to implement and experiment with NAMO algorithms, and build upon them in the next chapters.

[23] Using the same implementation mentioned in Section III. 2.1.2, that seamlessly falls back to a Breadth-First Search if a constant cost action space is detected.

III. 3 A new NAMO simulator

Now that we have introduced NAMO in Chapter II, and more specifically, the algorithms we build upon, their requirements, and the improvements we have added over them in the previous sections, it is now time to discuss their execution context. As mentioned at the beginning of this chapter, while the NAMO literature is rather well-endowed in terms of algorithms, no simulation tooling (e.g. source code or binaries) nor data (e.g. computer-readable environment description, parameters values, or raw unaggregated performance metrics) are ever provided. We had to derive our open implementations and datasets from the pseudocode when available, textual observations, images and occasional video footage. The required efforts encouraged us to design our simulator in accordance to three main principles: **interoperability**, **reproducibility** and **flexibility**. As a first step in this direction, we have made this simulator (and associated data) we will discuss in this section freely available at:

`https://gitlab.inria.fr/brenault/s-namo-sim`

III. 3.1 Overview of the simulator

FIGURE III.20: Live simulator view in Rviz in a map of our CITI laboratory. The robot (blue disc) reached its goal (dark blue circle) after moving a movable obstacle (outlined yellow rectangle with blue circle marking the robot configuration at release time) in a low-cost zone (coloured background is the combined costmap described in Chapter IV). Black shapes are static obstacles, and the path followed by the robot is traced as a blue (transit) and red (transfer) line. Pink cells are unaccessible (inflated obstacles).

Our state of the art in Chapter II revealed that NAMO is first and foremost treated as a 2D motion planning problem; although, the world's physical models used in NAMO literature still vary wildly from one approach to another. From various grid definitions to continuous

geometries, and with different modeling approaches for robot-to-obstacle motion dynamics, the need for a custom-tailored and flexible simulation tool naturally arose, in order to be able to experiment with even just the two algorithms described in this chapter. Figure III.20 gives you a small glimpse at how both these continuous and grid-based world models can cohabit in our simulation tool.

Before delving deeper into the design of this simulator, we would like to draw your attention to the following simplified Architecture Diagram in Fig.III.21, highlighting its main modules and data flows, from left to right. As can be seen in this diagram, world models (comprising both physical - mostly geometrical - and semantic data such as obstacle type) and simulation parameters (e.g. random number generator seed, NAMO algorithms parameters, ...) are loaded from data files through the **data import interface** into the **main simulation module**.

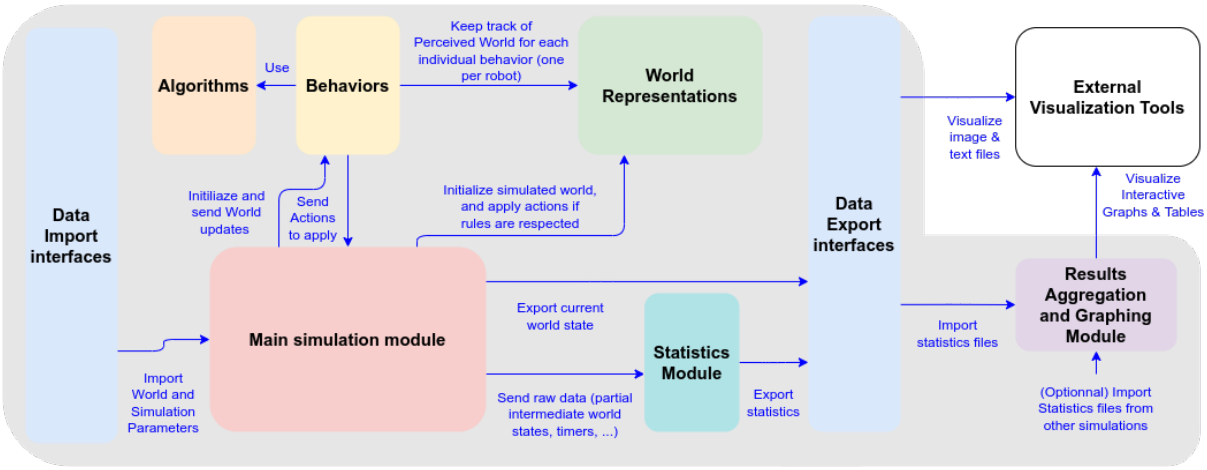


FIGURE III.21: Simplified Simulator Architecture Diagram - Solid colored boxes are simulator elements

This **main simulation module** is in charge of initializing the reference simulated **world representation** from the imported data, and depending on the simulation parameters, associating each robot representation in the world with an individual **behavior**. Each behavior is basically an object that keeps track of the perceived world knowledge of its associated robot, calls for planning **algorithms** to reach the goals that are given to it and executes the plans they yield, to try and actually reach said goals. Once the **main simulation module** is done initializing, it enters the main simulation loop.

Each loop consists in first informing all the behaviors (i.e. robots) of the reference simulated world state (and the behavior updates its perceived world knowledge according to its sensing parameters). Then all behaviors are asked for their next discrete action. Finally, the main simulator loop updates its simulated world model according to its constraints (e.g. refusing to apply a robot manipulation action if the obstacle is not actually movable for the robot), guaranteeing the world state to remain valid.

Until all behaviors return that they have finished executing all their goals, the main simulation loop collects and saves raw execution data (e.g. actions, their results, changes to the reference world state, ...). Once the loop is exited, the **main simulation module** hands the raw

execution data to the **statistics module**, that generates relevant statistics either relative to the overall world state or to each robot. These statistics are exported to a file, as well as the final world state in another file (that may be reused as an input world model for another simulation) through the **data export interface**.

Both statistics file and world state file can be visualized as is, using **external visualization tools** (e.g. any Web Browser, Inkscape, the system's default image viewer, any text editor...). The statistics file can also be handed over to the Results Aggregation and Graphing Module (in purple), that will, as its name suggests, summarize the data into paper-printable LaTeX tables, and generate interactive HTML graphs that allow to explore the evolution of the various collected statistics in depth. A Robot Operating System (abb. ROS) export layer is also available to allow live visualization of the world & behavior state (as shown in Fig.III.20).

As a final general comment on the simulator's architecture, it should be mentioned that it was fully written using the Python programming language. This choice was motivated by the widespread use of the language both in the robotics community[24] and the general public[25] - which facilitates both its reuse by third-parties and our use of high-quality community libraries. The code was written as to accommodate both Python 2 and 3, ROS1 and ROS2, in order to make it usable on both legacy and new systems[26].

Now that an overview of the simulator's architecture has been given, the following subsections will serve to provide further insight into specific components. These details are themselves synthesized at the end of this section in a more elaborate Architecture Diagram (Fig.III.29).

III. 3.2 World representations

Let us first clarify what we mean by "world representation": it is first and foremost a data structure storing physical (in our case, essentially geometrical) and semantic information about the environment's state. This data structure is constrained by a set of rules (functions) that define valid states and transitions between these states.

The simulator was designed to allow **flexibility** as to the world representation used either as the reference simulated world W_{ref} or the individual perceived world associated with each behavior $< W_{R_1}, \ldots, W_{R_i} >$. This flexibility allows for trade-offs between computational performance and expressivity[27] of the experiments.

Baseline Data Structure In all the scenarios presented in the figures that appeared until now, and in all the experiments described in the following chapters, both the reference simulated world W_{ref} model and the individual perceived world associated with each behavior

[24] Largely due to it being officially supported by the de facto standard Robot Operating System.

[25] Python is the most popular programming language since the last quarter of 2021 according to the TIOBE index [129] and second quarter of 2018 according to the PYPL index [130].

[26] Despite the official deprecation of Python 2 since January 1, [131], legacy systems such as the many robots available in our lab (e.g. Pepper, Turtlebots 2, ...) are still tightly bound to Python 2 / ROS 1 because of a lack of constructor-backed support.

[27] By expressivity, we mean the ease of creation of new scenarios/problems that are relevant to our subjects of study in the following chapters.

$< W_{R_1}, \ldots, W_{R_i} >$, rely on a polygonal world representation like (Stilman, 2005)'s original paper [3]. As shown in Fig.III.22, this world representation is defined by a set of Entities, abstract objects with a unique identifier and a non-deformable continuous geometry - a polygon.

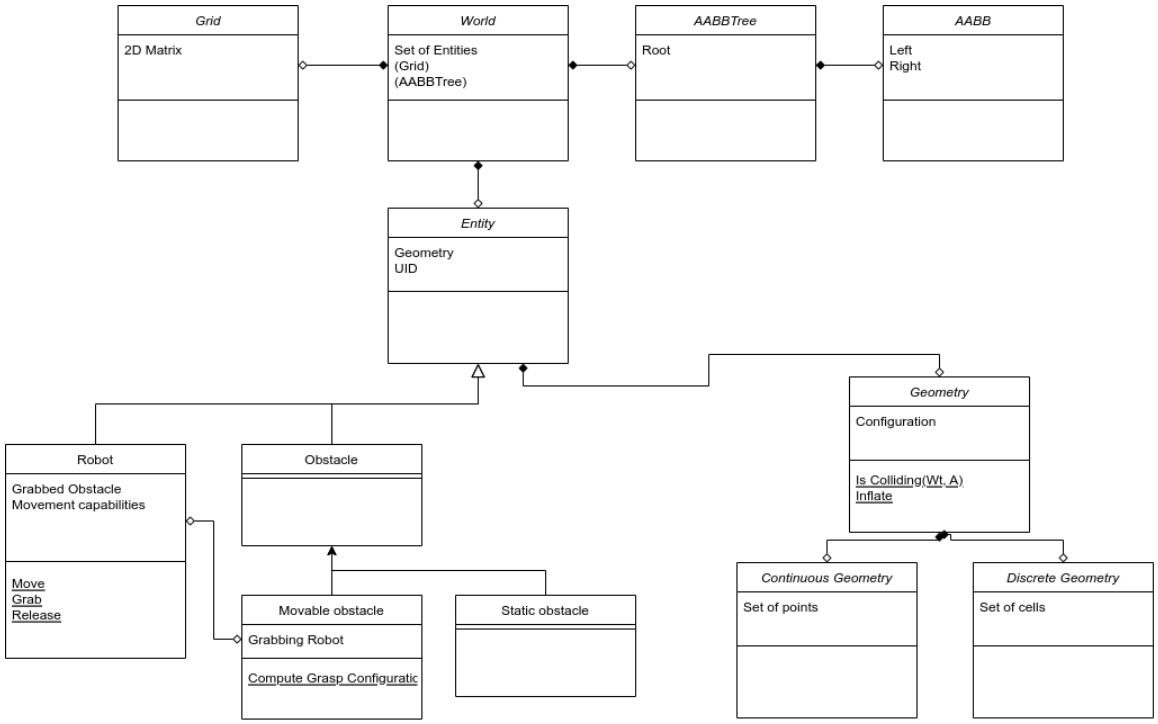
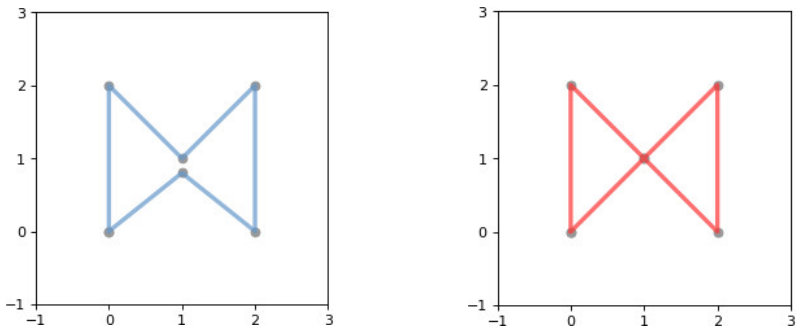


FIGURE III.22: UML Diagram of our main world representation, showing the attributes, operators and relations between the different data structures involved

This polygonal representation is provided by an external library called Shapely: a Python wrapper around the standard GEOS C/C++ library for computational geometry, that solely implements non-curved geometries[28]. More exactly, we use a subset of polygons: linear rings (non-self-intersecting polygons, illustrated in Fig.III.23), which can be either convex or concave.



(A) Valid Linear Ring (B) Invalid Linear Ring

FIGURE III.23: Valid and invalid examples of GEOS Linear Ring (non-self-intersecting polygon)

[28]The GEOS library implements the OGC Simple Features geometry model and provides all the spatial functions in that standard as well as many others. GEOS is at the core of widely used geospatial software like PostGIS, QGIS and GDAL.

Entities are divided into subclasses corresponding to the original model described in Section III. 1.1 : Robot, Movable Obstacle and Static Obstacle. This allows to unambiguously classify any entity at planning time, which is essential for NAMO algorithms.

Navigation Action Space $\mathcal{A}_N$ & Operators In Section III. 1.2.1, we reminded the two most primitive operators that can be applied to solve a NAMO problem: *Navigate* and *Manipulate*. We implemented *Navigate* as two distinct operators:

- **Translations** of any direction and any norm:

$$\mathcal{T}(W^t, R_i, x_{offset}, y_{offset}) = W^{t+1} \quad \text{with translation vector } x_{offset}, y_{offset} \in \mathbb{R}^2$$

- **Rotations** of any angle around the robot's geometric centroid $(x^t_{R_i}, y^t_{R_i})$:

$$\mathcal{R}(W^t, R_i, \theta_{offset}) = W^{t+1} \quad \text{with rotation angle } \theta \in [-2\pi, 2\pi]$$

Actual actions $A_N \in \mathcal{A}_N$ are instantiated *Translate* and *Rotate* operators with fixed translation and rotation parameter values. These values must be chosen so that the action space yields a finite robot configuration space, so that the underlying graph search algorithms (A^*, Dijkstra) may keep their completeness and optimality properties. In all experiments so far and in the next chapters, we use two discrete action spaces that yield navigation movements on a 2D square grid (of resolution $r = 0.1m$), including diagonal movement:

- One supposes the robot has an omnidirectional drive robot base:

$$\mathcal{A}_N^{omni} = \begin{cases} \forall \mathcal{T} \text{ with } (x_{offset}, y_{offset}) \in \{(0,r),(0,-r),(r,0),(-r,0),(\sqrt{2r},\sqrt{2r}), \\ \quad (-\sqrt{2r},\sqrt{2r}),(-\sqrt{2r},-\sqrt{2r},(\sqrt{2r},-\sqrt{2r})\} \\ \\ \forall \mathcal{R} \text{ with } \theta_{offset} \in \{\frac{\pi}{4}, \frac{-\pi}{4}\} \end{cases}$$

- The other supposes a differential drive robot base:

$$\mathcal{A}_N^{diff} = \begin{cases} \forall \mathcal{T} \text{ with } (x_{offset}, y_{offset}) \in \begin{cases} \{(r,0),(-r,0)\} \text{ if } \theta^t_R \in \{0, \pi\} \\ \{(0,r),(0,-r)\} \text{ if } \theta^t_R \in \{\frac{\pi}{2}, \frac{-\pi}{2}\} \\ \{(\sqrt{2r},\sqrt{2r}),(-\sqrt{2r},-\sqrt{2r})\} \text{ if } \theta^t_R \in \{\frac{\pi}{4}, \frac{-3\pi}{4}\} \\ \{(-\sqrt{2r},\sqrt{2r}),(\sqrt{2r},-\sqrt{2r})\} \text{ if } \theta^t_R \in \{\frac{-\pi}{4}, \frac{3\pi}{4}\} \end{cases} \\ \\ \forall \mathcal{R} \text{ with } \theta_{offset} \in \{\frac{\pi}{4}, \frac{-\pi}{4}\} \end{cases}$$

Manipulation Action Space $\mathcal{A}_M$ & Operators In Section III. 1.2.3, we reminded the three main manipulation action space classes used in NAMO: *Grasping*, *Pushing* and *Manipulation Primitives*. We only implemented the *Grasping* class manipulation action space for the primitive *Manipulate* NAMO operator. More specifically, we defined our action space $\mathcal{A}_{\mathcal{M}}$ with four generic operators:

- A **Grasp** operator that fully constrains the obstacle's movement (i.e. robot and obstacle "become one", and all translations and rotations are applied at the center of the robot), by creating a static transform between the grabbing robot R_i and a single grabbed obstacle M_j at the grasping point G_k (Cf. Fig.III.2b) - which can of course only happen if neither the robot nor the obstacle are already respectively grabbing or being grabbed by another entity:
$$Grasp(W^t, R_i, M_j, G_k) = W^{t+1}$$

- A **Release** operator that removes the static transform between the grabbing robot R_i and its grabbed obstacle M_j at the grasping point G_k (only valid if the transform exists in W^t, of course):
$$Release(W^t, R_i, M_j, G_k) = W^{t+1}$$

- Similarly to the navigation action space $\mathcal{A}_N$, **Translations** of any direction and any norm:
$$\mathcal{T}(W^t, R_i, M_j, G_k, x_{offset}, y_{offset}) = W^{t+1} \quad \text{with } x_{offset}, y_{offset} \in \mathbb{R}^2$$

- And again, similarly to the navigation action space $\mathcal{A}_N$, **Rotations** of any angle around the robot's geometric centroid (x_R^t, y_R^t):
$$\mathcal{R}(W^t, R_i, M_j, G_k, \theta_{offset}) = W^{t+1} \quad \text{with } \theta \in [-2\pi, 2\pi]$$

In all the experiments so far and in the next chapters, the parameterized translation and rotation operators are slightly different for the manipulation action spaces $\mathcal{A}_M^{omni}$ and $\mathcal{A}_M^{diff}$, as they assume 60° rather than 45° rotations - this is done to reduce the manipulation search space to accelerate computations, while maintaining good maneuverability. Also, in order to allow the robot to start planning when grasping an obstacle (which is useful if its plan is interrupted while manipulating), the *Release* operator is added to the navigation action space $\mathcal{A}_N$.
We chose to only implement a *Grasping* class action space for two main reasons:

- since **we preserve the original hypothesis from both (Stilman, 2005) and (Wu&Levihn, 2014)'s algorithms that, a single robot may only move a single obstacle at the same time**, the Grab and Release operators it requires for the robot allow for non-ambiguous modeling of movable obstacle reservation (Cf. Robot and Movable Obstacle in Fig.III.22);

- the static transform created by the grasp between the robot and the obstacle's motions makes it easy to guarantee that **the robot action space will yield a finite configuration space for both the robot *and* the obstacle**. As explained before, having a finite (robot,

obstacle) configuration space guarantees a finite manipulation search graph for the algorithms used for motion planning (A^*, Dijkstra), which is a requirement for them to keep their completeness and optimality properties.

The original implementation of (Stilman, 2005)'s algorithm [3] actually used *Manipulation Primitives* based on forward-simulation of obstacle dynamics through force application upon a single grasp point on the obstacle. In this original action space, while the robot was constrained to translational accelerations during manipulation, the obstacle could rotate around the grasp point. This rotational slip made it possible to solve cases such as the one in Fig.III.24. However, this model required additional physical data for each movable obstacle (*center of mass*, *mass*, *moment of inertia* and *viscous friction parameters*), which was not actually provided and disputably acquirable by a robot in a real world environment. As this model also did not yield a finite configuration space that would preserve the completeness of the algorithm, we chose not to implement it.

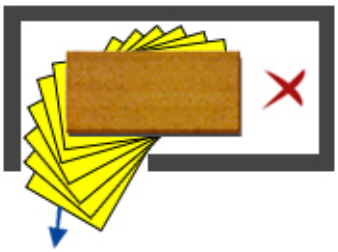


FIGURE III.24: A large, constrained object requiring manipulation motions that consider dynamics (copied from [3]). Red cross is the goal, blue circle and arrow are the grasping point and the force vector.

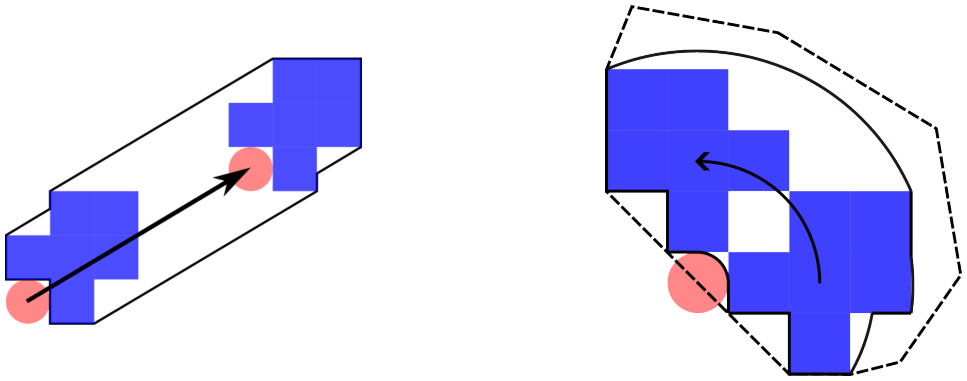
On the other hand the first implementation of (Wu&Levihn, 2014)'s algorithm used a *Pushing* class action space where the robot's motions were limited to axis-aligned translations in the direction of the pushed obstacle. The later improved implementations of this algorithm [34, 7] however dropped this action space in favor of a *Grasping* action space still limited to axis-aligned translations, but that already allowed the resolution of a much wider array of problems. This comforts our own choice to only study a *Grasping* class action space.

Continuous Collision Detection In order to keep the simulation's and the robots' world representations valid as they compute and execute plans, these operators must be valid non-colliding actions. As specified in Section III. 1.2.1, these operators describe a *continuous* contact-free motion of the object. Then, for these operators to be valid, no other entity may intersect with the swept area $S(q_R^t, q_R^{t+1})$ of the robot's shape as it translates or rotates between times t and $t+1$:

$$S(q_R^t, q_R^{t+1}) \cap W_{/R}^t = \emptyset$$

Efficiently computing this swept area for polygons is non-trivial, especially in the case of rotations, as the vertices' curved trajectories create a non-polygonal area (Cf. Fig.III.25). In order to approximate this area, we implemented Foisy et al.'s "Safe Swept Volume Method

for Collision Detection" [132] used by Stilman in his thesis work [19]. As the name suggests, this model is collision-safe, as it either finds the exact swept area (Fig.III.25a) or an overestimate (Fig.III.25b): thus making it impossible to fail collision detection (false negatives). False positives are however possible, although they can be reduced by decomposing the considered action into smaller ones, yielding a tighter estimate of the swept area.



(A) Arbitrary direction & norm translation. Real swept area needs not be approximated (Cf. Appendix A).

(B) Counter-clockwise 90° rotation. Black line is real swept area, dotted line is Foisy's approximation.

FIGURE III.25: A circular robot (red) manipulates a polygonal obstacle (blue), sweeping an area delimited with a (black) line that needs to be free of obstacles.

As efficient as this continuous collision detection strategy may be, it would still be computationally expensive to use it at every planned or executed movement, for both the robot and obstacle geometries. Also, checking for intersections with every other entity in the environment, regardless of their proximity to the action would remain too expensive. Hence, we use two data structures representing the world's geometry, in order to minimize the number of full continuous collision checks using Foisy's method:

- **Incremental Binary Occupancy Grid** We implemented a grid model that Stilman created for his algorithm [3], in order to accelerate robot collision detection in particular. This model consists first in creating an integer 2D matrix - or planar grid - originally filled with zeros. Then all the world's entities polygonal perimeters are rasterized, after being inflated by the robot's circumscribed circle's radius, and each cell of the rasterized polygons is incremented by one in the grid (Cf. Fig.III.26a). Upon movement of an entity, the initial state rasterized cells are decremented by one in the grid, while the new state cells are incremented by one (Cf. Fig.III.26b) - allowing for fast updates of the grid keeping its memory footprint reasonable. Thanks to this grid, robot collision checks can be done by verifying whether the center cell of the robot has a 0-value in the grid, and only if it is non-zero, then use Foisy's method to complete the check. This drastically accelerates all searches, and in particular, searches over the navigation action space $\mathcal{A}_N$.

- **AABBTree** As to avoid checking for intersections with every single other entity in the world, we applied a common technique used in state-of-the-art physics libraries: an Axis-Aligned Bounding Boxes Tree [133]. It basically consists in simplifying and grouping the world's geometries as axis-aligned rectangles in a tree structure. Leaves are actual

entities' bounding boxes, and branch nodes are encompassing bounding boxes that do not overlap within the same tree depth. This allows to quickly assess which entities are close enough from the swept area geometry we want to check for intersection with, and require a collision check with the actual entity's geometry instead of its bounding box.

(A) (B)

FIGURE III.26: (Stilman, 2005)'s incremental binary occupancy grid model. (A) Initial state, (B) State after the rounded obstacle is moved to the right and bottom.

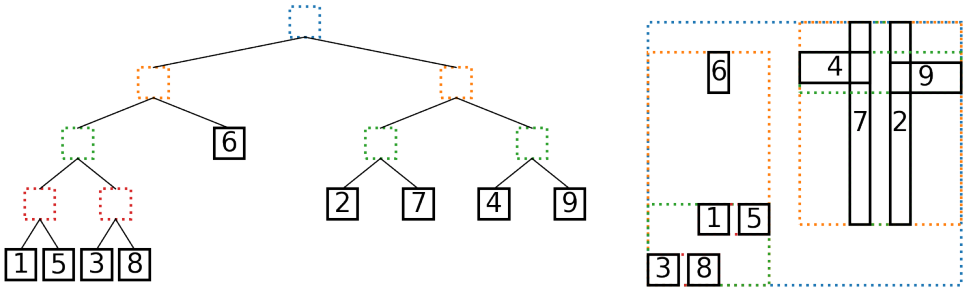


FIGURE III.27: AABBTree example - Axis-Aligned Bounding Boxes on the right, tree structure on the left. Leaves are numbered by insertion order.

All in all, collision detection is assured with these three complementary data structures for efficient and precise collision detection: polygonal overestimates of swept areas, the AABBTree and the incremental binary occupancy grid.

III. 3.3 Main simulation module

As stated in the previous chapters, and in the very title of this thesis work, one of the end goals is to have *multiple* robots solve NAMO problems in a common environment. The main simulation module is tasked with the job of initializing both the reference simulated world model W_{ref}, and the individual robot behaviors that interact with it (and their individual perceived world models W_{R_i}) as illustrated on the left in Fig.III.28.

As shown in Chapter II, existing NAMO algorithms are only designed to compute valid actions (that is, discrete physical movements) to execute for a single robot. Hence, when it comes to executing the individual actions produced by each robot's behavior in each simulation loop, a problem arises: how do we execute these multiple actions? For the sake of **reproducibility** and ease of implementation & debugging, we chose to build our simulation loop so that:

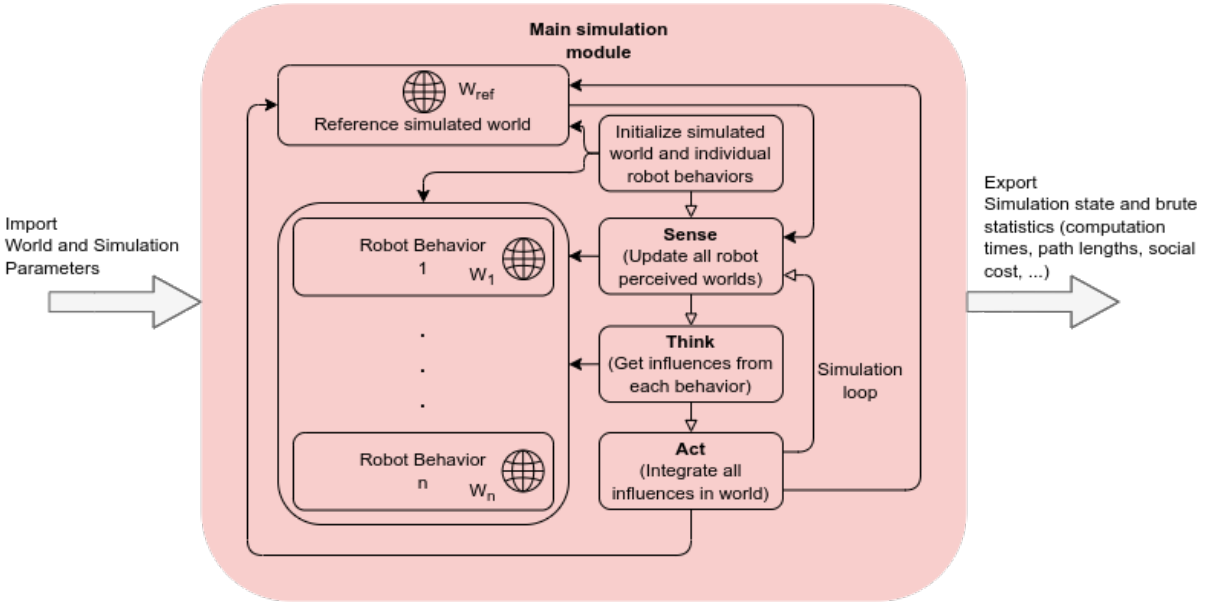


FIGURE III.28: Main simulation module

- **all robots are synchronized with the simulator's clock**: that is, at each simulation time step t, each robot is informed of the reference world state W^t and asked to compute and execute a discrete action for this time step[29].
- **actions are applied following Jacques Ferber's influence-reaction model [134]**: that is, the simulator considers these actions returned by each robot's algorithm to be *influences* (i.e. action attempts) instead of unquestionable orders. These *influences* need to be integrated by the reference simulated world representation, according to its internal constraints, into a *reaction* that will synthesize the actual change to be applied to transition the world state from W^t to W^{t+1}. Basically, if two robots were to try to exert two incompatible influences (e.g. manipulate the same obstacle at the same time), the *reaction* would prevent the execution for both and inform the two robots of the failure - while executing all other compatible influences. This way, the world model can be kept consistent, while allowing multiple actions to happen simultaneously, and guaranteeing an outcome that remains independent of the order in which robot computes its action first.

This simulation loop is illustrated in Fig.III.28. It should be noted that in the same way that influences are computed for all robot behaviors (**Think**), before integrating them all at once in the reference simulated world (**Act**) following the pre-named influence-reaction model, we apply the same concept for sensing (**Sense**). This leaves room to implement the sharing and integration of the robots' perceived worlds, in a way that guarantees consistency at the planning (**Think**) stage.

[29] In a real world setting, this hypothesis would be verified as long as the constant time step for each discrete action is big enough to compensate for internal clock error and drift - and the robot's planning algorithm is guaranteed to yield an action that will yield the next world state W^{t+1}.

III. 4 Conclusion

Having presented our formalism in Section III. 1, the algorithms we will base our subsequent work on in Section III. 2, and their implementation in our open source simulation tool in Section III. 3, we synthesized all these elements in a Detailed Simulator Architecture Diagram in Fig.III.29. This figure summarizes the contribution that is our simulation tool itself, and provides the necessary context to understand how we formulated our Social and Multi-Robot extensions to the NAMO problem, as well as the solutions we propose in each case, which shall be presented in following chapters.

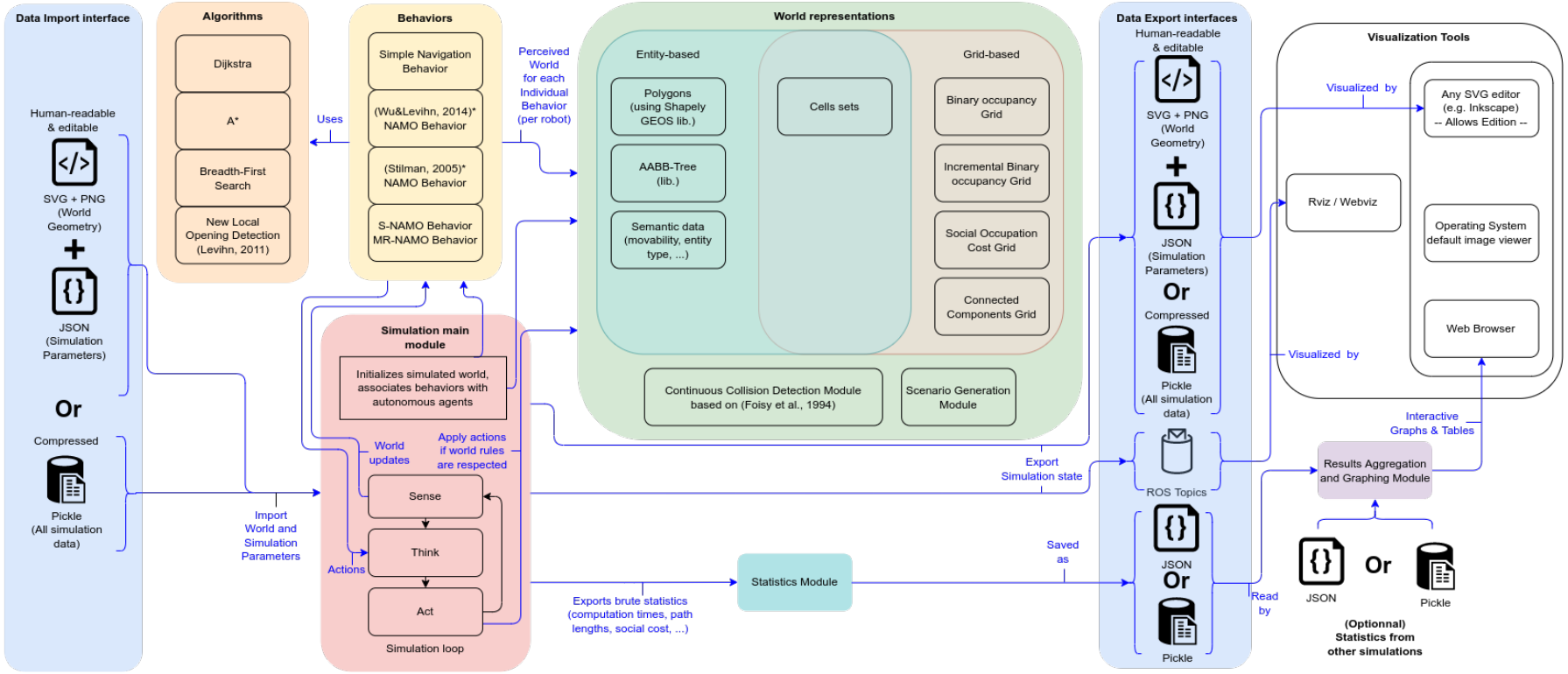


FIGURE III.29: Detailed Simulator Architecture Diagram - Solid colored boxes are simulator elements; transparent boxes are external tools. To facilitate reading, we recommend going back to the description of its simplified counterpart in Section III. 3.1.

Chapter IV

Socially-aware NAMO

IV. 1 Introduction - the General Socially-Aware NAMO Problem

We concluded our NAMO state of the art in Section II. 1.10 with the observation that **social concerns have never been addressed in NAMO problems**; although human environments are a clear target application for NAMO algorithms in the literature. Another important conclusion of this state of the art, is the highlighting of *obstacle placement choice* as a characteristic sub-problem of NAMO, compared to neighboring problems[1]. We have found that the only obstacle placement heuristic until now was to **move the object just enough to make an opening for the robot**. However, only minimizing the robot's displacement cost, without regards for other (potential) agents or its future self is bound to cause socially undesirable situations. One such situation is illustrated in Fig.IV.1a and IV.1b, using (Stilman, 2005)'s algorithm[2] presented in Section III. 2.2. Here, the robot's selfish choice would cause a human coming from the topmost doorway to get blocked; even in the absence of a human, the robot would still have hampered any future navigation towards the topmost or rightmost entryways for itself.

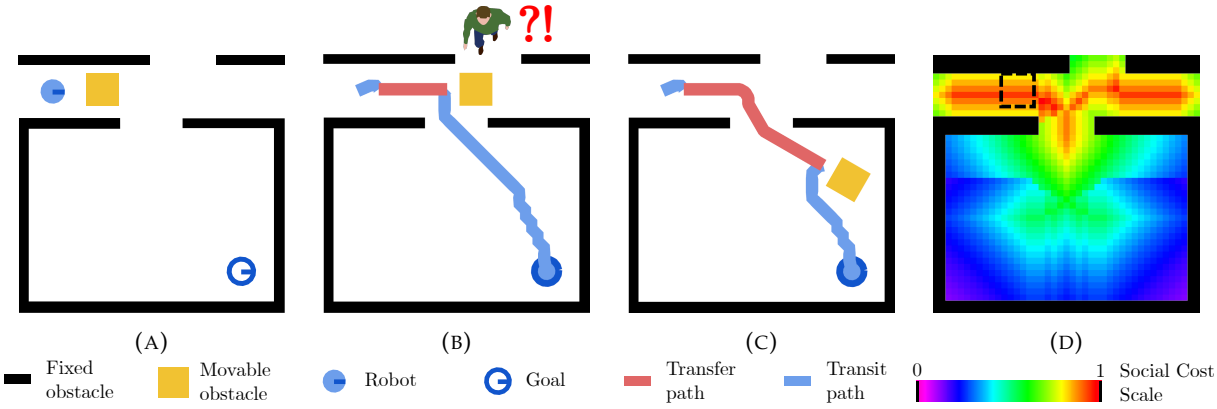


FIGURE IV.1: Basic socially-pathological NAMO problem. (A) The robot's goal is in the adjacent room but unreachable due to a movable obstacle. (B) (Stilman, 2005)'s NAMO algorithm solution. (C) Our social (S-NAMO) approach's solution. (D) Our social occupation costmap used by our S-NAMO approach to bias obstacle transfer to a more socially-acceptable location.

[1] As a reminder, NAMO neighboring problems are: Navigation Planning, Manipulation Planning, Rearrangement Planning, Assembly Planning, and combined Task and Motion Planning.

[2] (Wu&Levihn, 2014)'s algorithm, presented in Section III. 2.1, would generate the exact same resolution-optimal solution, as would all other NAMO algorithms, because of the same sole optimization of displacement cost.

Our exploration of Social Navigation (Cf. Section II. 2.1.1) reminded the need to have the robot's path planning respect social rules and conventions, as to improve not only safety, but also comfort, naturalness and sociability. The example scenario above goes to show that similar constraints must exist and be respected in the resolution of NAMO problems. From this observation, and our NAMO problem definition, we can now propose a general definition of the Socially-Aware NAMO problem:

General Socially-Aware NAMO Problem (abb. S-NAMO): The Socially-Aware NAMO problem adds to the General NAMO Problem (Cf. Section III. 1.7) **a set of social constraints $\mathcal{C}$ that can forbid certain actions A^t or affect their cost function $\mathcal{C}(A^t)$, depending on the workspace's state W^t.**

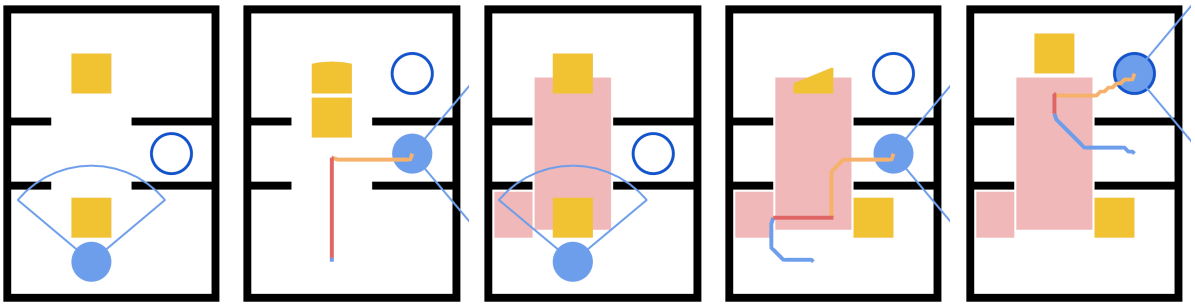
We also discussed in our Social Navigation exploration how these constraints are often expressed through the notion of social spaces. We then underlined how all these social spaces are only *temporary*, and only exist in the actual presence of humans at the time of plan computation and execution, at the exception of affordance spaces, that are generated by the *potential activity offered by objects*. In Social Navigation, adapting the robot's path planning only matters in the actual presence of humans in the environment. However, in a NAMO problem, since moved obstacles stay in place after they are manipulated, it is essential to find and apply social constraints (i.e. social spaces) that exist even in the absence of humans. Otherwise, as illustrated in the above scenario, humans will eventually be disturbed when they appear in the environment. That is why, in this chapter, we focus on expressing social constraints for S-NAMO based on the concept of affordance spaces.

In the following sections, we first present a naive local and binary affordance space model and discuss its application to (Wu&Levihn, 2014)'s algorithm. We then explore more complex affordance space and constraints models from the literature that could be applied to S-NAMO. This exploration leads us to the formulation of the "Social Placement Choice NAMO Problem", and our own affordance space model we call "Social Occupation Cost" (illustrated in Fig.IV.1d), and its application to (Stilman, 2005)'s algorithm (illustrated in Fig.IV.1c).

IV. 2 A first naive, local and binary constraint model

Our first approach to modeling a constraint based on the idea of affordance spaces was a naive, binary and local model of **taboo zones**, that we introduced in our first published paper in the 2019 Robocup Symposium [30]; that is, a set of user-drawn polygonal areas T, representing areas of interest to humans (e.g. display space, carpet, corridor, crossing, ...). When planning, the robot is required to ensure that the final state M_i^g of any manipulated obstacle M_i during the execution of plan $\mathcal{P}$ does not intersect with any of the taboo zones, formally:

$$\forall M_i \in \mathcal{P}, M_i^g \cap T = \emptyset$$



(A) (B) (C) (D) (E)

FIGURE IV.2: Simulation of a two-goals scenario with NAMO (A, B) vs. S-NAMO (C, D, E). Initial world state (A, C). Robot as blue disc with conical Field of View, movable obstacles in yellow, unmovable walls in black, goal as dark blue circle, taboo zones in red. Blue line is transit path to obstacle, red line is transfer path, orange line is transit path to goal.

We implemented this simple model into (Wu&Levihn, 2014)'s algorithm, presented in Section III. 2.1, by adding the above constraint to its obstacle manipulation planning routine (Algorithm 4, l.19), and ran experiments to explore the concept. For instance, take the simple scenario illustrated in Fig.IV.2a: two rooms separated by a corridor, each doorway being blocked by a movable obstacle. The robot with a conical limited field of view must first reach a goal in the right part of the corridor, to then be informed of a second goal in the top room once the first one has been reached. At the initial state shown in Fig.IV.2a, the robot has already been informed of the walls and can only sense the first obstacle. Using (Wu&Levihn, 2014)'s algorithm, the robot reaches the first goal by pushing the obstacle forward as it advances toward the goal, only seeking to minimize its own displacement cost, discovering part of the top-room obstacle along the way, as illustrated in Fig.IV.2b. By doing so, however, the robot cannot compute a plan to its second goal using (Wu&Levihn, 2014)'s algorithm, as it does not allow the manipulation of more than one obstacle per plan (and here, the plan would require two). In Figure IV.2c, the same initial situation is exposed, but manually adding two taboo zones: a big one to indicate the doorways, and a small one to indicate the presence of a screen on the wall. These taboo zones force the robot to move the bottom movable obstacle to the right rather than push it along (Cf. Figure IV.2d), making it possible for the second plan to be computed and goal to be reached (Cf. Figure IV.2e).

While this simple approach is indeed capable of yielding some useful results, as we have just seen, it however showcases some non-negligible drawbacks:

- **Manually inputted data:** As previously stated, the method currently needs manual user input to draw taboo zones in relevant spaces, which simply does not scale. The whole point of a robotics algorithm is to have the robot *autonomously* take decisions for itself using its own sensing data - **thus it should determine affordance spaces by itself**;

- **Solution Space Destruction:** By completely forbidding the robot from placing obstacles in the taboo zones, the problem's search space is artificially reduced (with the positive side effect of reducing computations), but *so is the solution space*. In a human environment, there could be many objects creating relevant affordance spaces around them: modelling

them all as taboo zones may very well cover the entire space with forbidden areas, making it simply impossible to solve NAMO problems at all. Also, each space may not be as important as the other to respect: **affordance spaces should be represented in a way that allows to prioritize not invading some area over some less important area**.

IV. 3 Object placement & affordance spaces in the literature

As we have just concluded in the previous section, a finer, more generalizable and scalable constraint model is needed for Social NAMO problems. Rather than diving head-first in the specification of a new and more complex model, let us take the time to review existing work.

As mentioned in Section II. 2.1.1, Lindner & Eschenbach were the first roboticists to formalize the concept of *Affordance Space* in [82], completing this formalization in later work [21, 32, 135, 20]. As a reminder, affordance spaces are defined as spaces related to a **potential** activity provided by the environment - in resonance with the concept of *Activity Space*, spaces associated with actions **being performed** by agents. Lindner & Eschenbach further decomposed affordance spaces (syn. "Affordance Regions") into sub-spaces (smaller "regions") as illustrated in Fig.IV.3. An object (here a picture on a wall) that provides an *affordance* to potential *agents* is called an *affordant*: the space occupied by the object is thus called the *Affordant Region*, while the space occupied by the potential agent(s) is called *Potential Agent Region*. The space between these two regions that connects them and allows for the potential interaction to exist is called *Potential Transactional Region*. Together, the potential agent region and the potential transactional region are called the *Potential Core Region*.

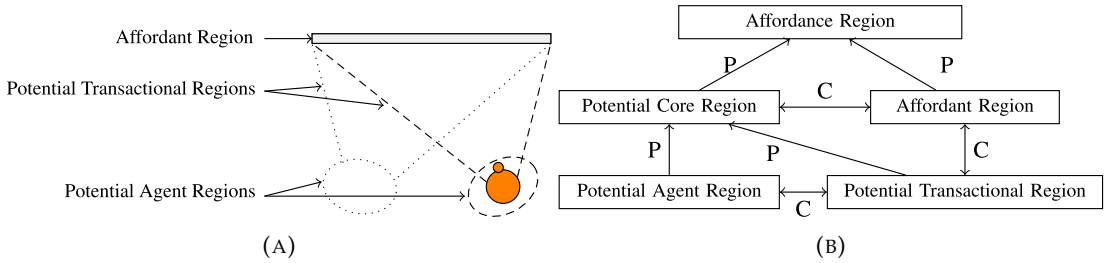


FIGURE IV.3: Figures from [20] illustrating their affordance space decomposition. (A) A picture with a viewability affordance, generating two affordance spaces: an unused one (dotted lines), and one being used (dashed lines) by a human (orange circle), i.e. an activity space. (B) Taxonomy of an affordance space's components. "P" indicates parthood while "C" indicates connection. If a region is part of another region, then these regions are also connected.

The geometry of these regions depends mainly on the afforded activity type, the spatial structure of the affordant object(s), and the abilities of the agent(s). That is why, while Lindner & Eschenbach did provide a solid theoretical foundation in their aforementioned works, notably in terms of taxonomy, they impart very few examples of actual affordance space geometries.

In [21], Lindner & Eschenbach presented a lab experiment with a PR2 robot, where its task was to find and navigate to a *socially best placement for recharging* in a room (Cf. Fig.IV.4a), using

manually designed affordance space models relevant to the experiment - designated as *affordance map* (Cf. Fig.IV.4b). Using this knowledge, the robot would sample candidate charging affordance spaces, and rank them using the concept of "socio-spatial reason". This concept associates each affordance with a positive or negative rank of entering it, with three degrees of strength (Strong, Medium or Weak). They finally use a pre-programmed decision rule to synthesize the social adequacy of intersecting multiple affordance spaces (Cf. Fig.IV.4c). This overall model addresses to some extent both drawbacks discussed in the previous section: affordance space instances are generated from pre-programmed models rather than manually drawn, and social reasons allow ranking between different possible choices instead of absolutely forbidding space invasion. **However, the affordance space models themselves are manually programmed, and are not trivially generalizable**. Also, regardless of the amount of overlap, the social adequacy is deemed to be the same: **the ranking of social adequacy is not proportional to the penetrated area**.

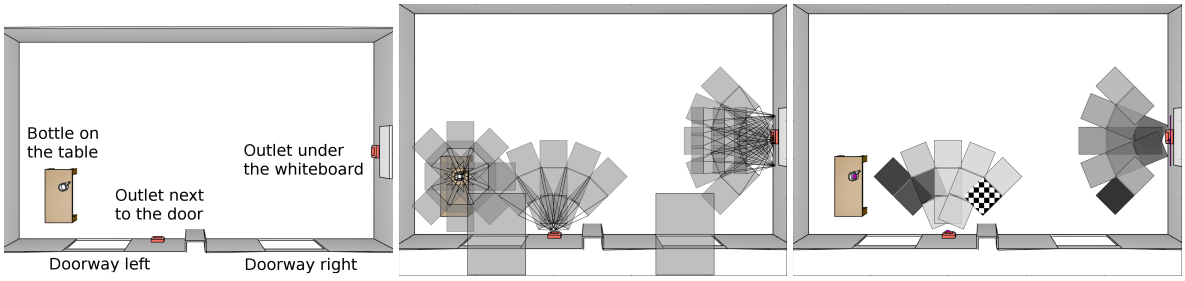
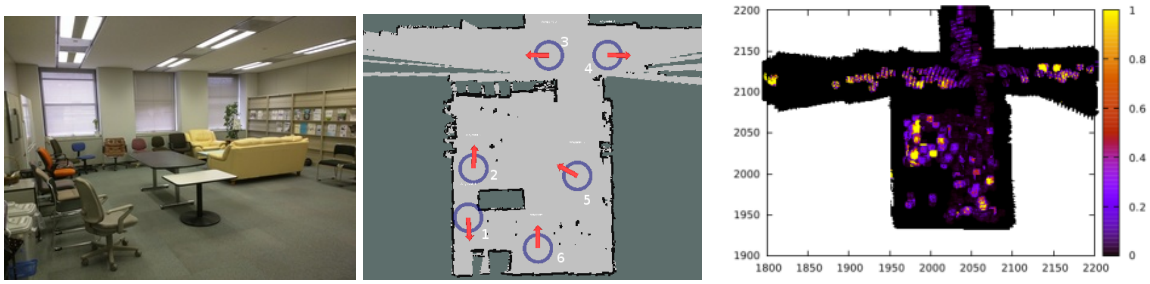


FIGURE IV.4: Illustration of Lindner & Eschenbach's lab robot experiment from [21]. (A) Initial state of the experiment's environment. (B) Affordance spaces corresponding to each affordant object in A. (C) Charging affordance spaces - the lighter, the higher social adequacy of using it. Checkered space is selected as best charging space for the robot.

Rather than inferring affordance space instances from the objects layout and manually programmed affordance models, Limosani et al. [22] **learned their affordance map from long-term real-world human activity observation**, using only an onboard 2D laser scanner. Said affordance map consists in a grid where each cell is associated with a probability of human presence, as illustrated in Fig.IV.5. While this allows for fully autonomous discovery of affordance spaces, Limosani et al.'s map is limited to the modeling of Potential Agent Regions, but not the Potential Transactional Regions nor the Affordant Regions. Also, generated affordance knowledge cannot be generalized to new environments; given how time-costly it is to generate it (70 days of cumulated observation time over a span of 4 months in their experiment), this is a significant drawback. Indeed, in a real world setting, the robot may not have the opportunity to observe that long, because of other tasks' timing requirements or privacy concerns.

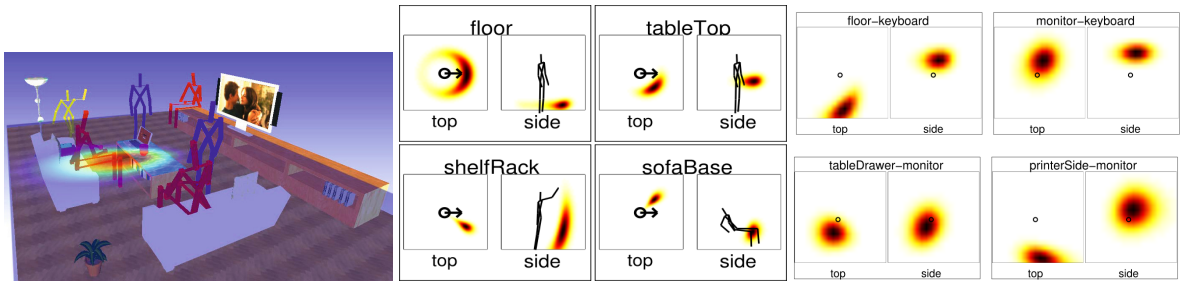
Rather than using real-world activity observation data to map affordance spaces, Jiang et al. learned affordance spaces through the analysis of manually-designed 3D environments and real-world 3D point clouds datasets [136, 137, 138, 139, 23] **without actual humans in said environments**. To do so, they have the robot "hallucinate humans", that is, randomly sampling *simulated humans* in the environment so that they could "interact" with existing objects in the



(A) (B) (C)

FIGURE IV.5: Figures from [22] showing Limosani et al.'s affordance space model in their coffee lounge test environment. (A) Test environment in which furniture may be moved by humans during the 4-month experiment. (B) Autonomous navigation waypoints for the observation sequence. (C) Cumulated affordance map after 4 months of daily observation, but only using the data corresponding to the days when the environment was in configuration A - the yellower, the higher the probability of human presence.

environment. This sampling is guided with pre-programmed constraints based on knowledge of how humans interact with objects and potential functions that model how a human pose is related to an object (e.g. no sitting on the top of a shelf, sitting at a certain distance from a television, ...) - in other words, Potential Agent Regions models (Cf. Fig.IV.6a). Contrary to Lindner & Eschenbach, Jiang et al. assume that one object type only yields one affordance, and that only one hallucinated human may use each object (one hallucinated human can still use multiple objects). Once relevant human poses have been sampled in the training environments, probability distributions of object poses around humans can be learned, producing a model of Affordant Regions (Cf. Fig.IV.6b) - Transactional Regions are again not modeled in this approach. In their latest paper [23], they further improve this concept by iteratively repeating the process, adapting their affordance models to sample ever more semantically relevant human poses in the scenario(s) at hand, yielding ever better Affordant Region models[3].

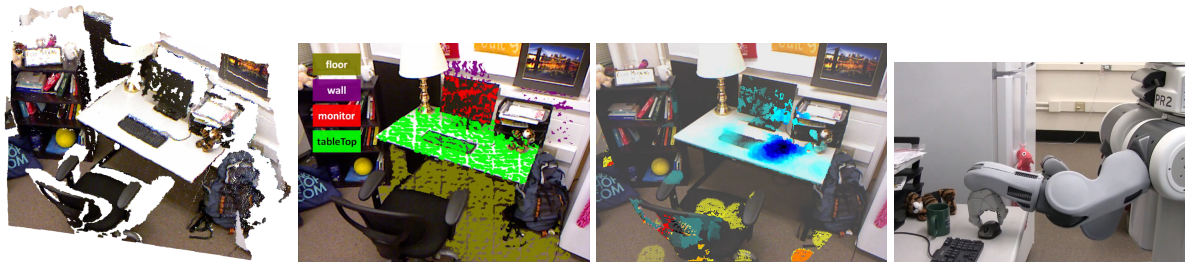


(A) (B) (C)

FIGURE IV.6: Illustrations of Jiang et al.'s affordance space model and its application to placement choice in a robotic rearrangement planning problem. (A) Sampled human poses based on objects' affordances - the potential function of the TV shown as a heatmap causes the sitting human pose to be sampled with high probability. (B) Examples of learned object-human affordant spaces, projected from top and side views for different object classes. The heatmap represents the distribution of the object in a 5 m $\times$ 5 m $\times$ 3 m space given a human pose facing right. (C) Examples of learned object-object affordant spaces, the heatmap for each object pair "obj1-obj2" shows the distribution of obj1 given obj2 at the center facing right.

[3]It is to be noted that the code and data used to be hosted on their now unavailable website http://pr.cs.cornell.edu/hallucinatinghumans/, but can still be accessed through the Internet Archive.

Coincidentally, one of the three main use cases of their model is to find the best placement for a given object in an existing environment, a variation of the rearrangement planning problem. Thus, the authors also show how ideal placements of objects can be determined with a similar process of sampling imaginary humans in the test environment, then sampling relevant placements of the object from the learned affordant region models relatively to these *hallucinated humans* (Cf. Fig.IV.7). The two other use cases of Jiang et al.'s models were *scene understanding* - the classification of pre-segmented 3D RGB-D point clouds, and *indoor scene synthesis* - populating empty 3D models of rooms with objects.



(A) (B) (C) (D)

FIGURE IV.7: Figure from [23], showcasing Jiang et al.'s real-world robot experiment where the robot must place a given computer mouse in the best possible placement. (A) RGB-D point-cloud of the environment. (B) Semantic segmentation using their algorithm. (C) Sampling of possible human poses (red heatmaps) and possible object placements (blue heatmaps).(D) The robot successfully places the mouse in the best location.

While Jiang et al. observe, even in their latest state of the art (2016, [23]), that the problem of computing relevant object placements has been barely addressed in the robotics literature, it is a core subject in the field of indoor scene synthesis - as shown in one of the latest surveys of the domain [140]. In order to generate realistic indoor environments, synthesis algorithms similarly need implicit or explicit rules to help decide of meaningful object placements. For example, the reference work of Yu et al. [141] derived a set of rules from expert knowledge found in the interior design literature [142, 143], and learned sensible parameter values from automated analysis of a set of human-designed environments. More recently, the popular work of Qi et al. [144], yielded consistently more accurate and realistic results than Yu et al.'s state-of-the art. They achieved this notably by borrowing Jiang et al.'s human-centric affordance modeling, and using a much larger human-designed indoor scene dataset as learning material.

Conclusion: The works we have discussed until now, and the wider indoor environment synthesis literature - that we shall not delve into here, certainly provide very relevant models for object placement constraints and space affordances, that could be used to guide our Social NAMO algorithms. **However, they all rely on the hypothesis that extensive and trustworthy semantic knowledge about the environment is readily available**, or that it is possible to observe real-world human activity long enough in the environment to derive affordance space knowledge. However, **acquiring such semantic knowledge in situ is no easy task for a robot**, as the active work in semantic mapping capabilities shows [76]. Also, **all these affordance space models we could find are inherently local**: the spaces they generate only cover

limited regions of the environment (as can be clearly seen in Fig.IV.4b, IV.5c and IV.7c). For environments with a low number of semantically identifiable objects, or when the robot fails to detect such objects, it would become impossible to reliably leverage such models to guide the robot in choosing good placements for manipulated obstacles in NAMO problems. Finally, we must insist that in NAMO, the main robot purpose remains *navigation*, that is, to reach the goal while optimizing the robot's displacement cost. To reach such compromise, the robot needs to be able to evaluate the social relevance of manipulating the obstacle to any configuration that helps with the robot's navigation - not just to find the best placement for the moved obstacle. All these reasons bring us to the conclusion that **a less object semantics-dependent, more global model of social adequacy for obstacle placement is needed**.

IV. 4 The Social Placement Choice NAMO Problem

Our exploration of the Social Navigation literature (Cf. Section II. 2.1.1) reminded that a common way to approach social space modeling is through the use of costmaps - grids where each cell is associated with a cost of entering it. The higher the cost, the higher the disturbance to humans. According to the surveys we have discussed, while this approach may oversimplify the problem, it allows for simple search, representation, and combination of many types of social considerations. Hence, we choose to formulate our Social Placement Choice NAMO problem using a **global costmap** that covers the entire environment.

For this, let us first define notations and structures of the problem, in the continuity of those given in Section III. 1:

- a discrete representation (syn. raster) $G(W^t)$ of workspace W^t as a 2D binary occupancy square grid, without loss of generality. Cells are addressed by tuples of coordinates $(x, y) \in \mathbb{N}^2$.
- another 2D grid G_{SC} built from $G(W^t)$ (thus sharing the same dimensions, position, orientation and resolution), where each cell is associated with a social occupation cost $SC(x, y) \in \mathbb{R}^+$.

We write $Cells(E_i^t)$ the cells occupied by any entity E_i at time t. The social occupation cost $SC(M_i^t)$ is written as the sum SC(x, y) of each cell occupied by a movable obstacle M_i at time t:

$$SC(M_i^t) = \sum_{\forall (x,y) \in Cells(M_i^t)} SC(x, y) \tag{IV.1}$$

Finally, we define the total social occupation cost of the NAMO plan $\mathcal{P}$ as the sum of the social occupation cost of all manipulated obstacles in their final configuration M_i^g:

$$SC(\mathcal{P}) = \sum_{\forall M_i \in \mathcal{P}} SC(M_i^g) \tag{IV.2}$$

With this formalization, we can now fully define the problem we shall address in this chapter as a multi-objective optimization problem:

The Social Placement Choice NAMO Problem (abb. SPC): Given an initial workspace configuration W^{t_0}, the Social Placement Choice NAMO problem consists in computing, if it exists, a plan $\mathcal{P}$ for a manipulator robot R from its initial configuration $q_R^{t_0}$ to a goal configuration q_R^g, while being allowed to *Manipulate* movable obstacles $M_i \in M$ as necessary, guaranteeing the absence of any collisions between the robot R and manipulated obstacles with any other obstacles $O_i \in O$, **while minimizing both the resulting displacement cost**[4] $\mathcal{C}(\mathcal{P})$ **and total social occupation cost** $SC(\mathcal{P})$ **caused by the final configurations** M_i^g **of obstacles it moves**.

IV. 5 The Social Occupation Cost Model

As we concluded in Section IV. 3, it is no trivial matter to acquire enough semantic knowledge about objects in the environment, their affordances and associated affordance spaces, to inform every cell in our social cost grid. Fortunately, we have found a very relevant insight in the field of interior design, that is well-summarized in the following quote:

> *"Providing adequate space for movement is essential and requires careful consideration of ergonomics, scale, and organizational flow"* Mitton and Nystuen, 2016 [142]

That is, the layout of human space is very much deliberate: as such, any free space in a human environment has, in and of itself, a strong semantic meaning we can use of *accessibility* and *circulation*. Free floor space, by offering an affordance of *Navigability/Accessibility* arguably generates an affordance space that spans its entirety. That is, in the terms of Lindner & Eschenbach [82], the entire free floor space is at the same time an affordant region, potential agent region and transactional region.

Incidentally, binary occupancy grids of fixed obstacles are common prerequisites of widely-used navigation frameworks like the ROS Navigation Stack [145], and most importantly, not only do they model the occupied space of fixed obstacles, *they also do in fact model free space*. Building such grids from real world observation is a straightforward and arguably mastered field, and they can even alternatively be obtained through fire evacuation maps almost systematically available in modern buildings [146].

IV. 5.1 Heuristic hypotheses

Following our argumentation above, let us assume that all free space in human environments can be considered as passageways that are *designed* to offer space for a certain number of people to walk abreast, which we will call *space allowance* (which holds especially true in public buildings [147]). Since we only assume to have the binary occupancy grid of static obstacles as prior knowledge, our focus can only be the preservation of the environment's original topology and space allowances.

To better understand how to do so, let us consider the schematized version in Fig.IV.8 of the basic pathological case previously introduced in Fig.IV.1 (repeated in Fig.IV.8a), where the

[4] typically expressed as energy, time, or distance

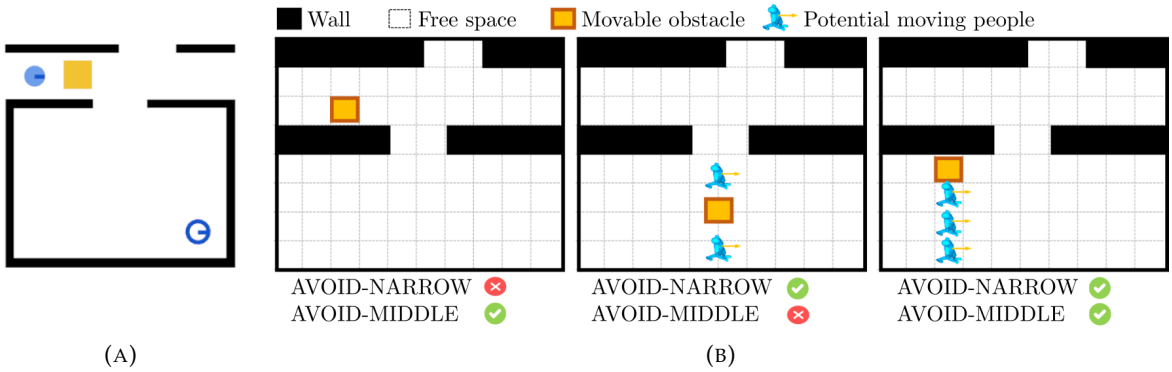


FIGURE IV.8: Illustrations of the AVOID-MIDDLE and AVOID-NARROW heuristic hypotheses (B) based on a schematized version of the basic pathological case presented in Fig.IV.1 (repeated in A).

human-sized grid decomposition highlights the impact of different object placement choices on the space allowance. When empty, the top corridor allows a minimum space allowance of 2 humans walking abreast (eq. 2 human units), while the bottom room allows for 4 people. As illustrated in the left-most figure of Fig.IV.8b, even leaving the obstacle closest to the wall would still halve the corridor's space allowance to a single human unit. On the other hand, as illustrated in the rightmost figures, leaving the obstacle in the room would either reduce the room's space allowance by half if left in the middle (2 over 4 human units), or by 25% if left closest to the wall (3 over 4 human units). Furthermore, in the "middle-of-the-room" case, the room is in fact split into two passageways, each only allowing a single human to pass abreast, instead of the initial 4, so the actual loss of initial space allowance could be considered to be of 75% in this case. Clearly, out of all options, leaving the obstacle in the largest space, and as far away from the middle of space as possible, appears to be the best choice to preserve the initial space allowance.

Thus, we propose the following **two heuristic hypotheses**:

1. AVOID-MIDDLE: The original free space allowance must be the least divided: thus, the cells in the "middle of the space" must have the highest local costs;
2. AVOID-NARROW: The narrower a passageway, the higher the relative loss of space allowance that can be caused by an object: thus, narrow spaces must have higher costs than wide spaces.

In the remainder of this Section IV. 5, we shall translate these hypotheses into mathematical operations, that will allow us to build a coherent social occupation costmap from the binary occupancy grid of fixed obstacles $G(F)$.

IV. 5.2 Computation steps overview

The algorithm we create to translate the above heuristic hypotheses can be summarized with the following four steps (which we shall delve further into in the following subsections):

1. **Identify the middle of space** cells, noted $SK(G(F))$ - that is, the thinnest discrete version (one cell wide) of the environment's shape equidistant to the obstacles boundaries, also called "skeleton" (Cf. Fig.IV.9.A.Top);

2. **Measure the space allowance** for each cell in $SK(G(F))$ - that is, the minimal distance from each cell to the fixed obstacles $\in F$ as to differentiate narrow and wide spaces (Cf. Fig.IV.9.A.Bottom);

3. **Convert** the previously determined minimal distance from obstacles into a social cost for each cell in the skeleton $SK(G(F))$, using a function f_{conv} that relates the distance to appropriate human-sized constants to ensure high costs in narrow spaces and low costs in wide ones (Fig.IV.9.B);

4. **Propagate** the social cost values from the skeleton cells in $SK(G(F))$ in a decreasing wave defined by a function f_{prop}, to ensure local highest costs in the middle of space (Fig.IV.9.C.).

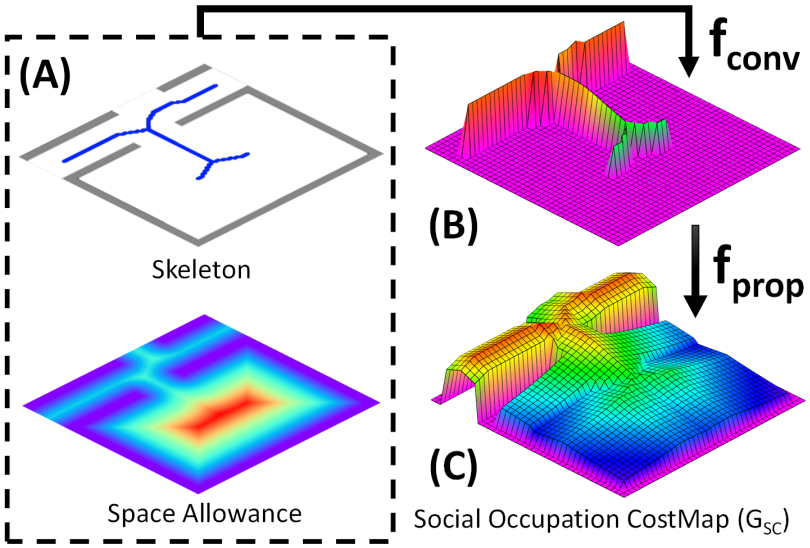


FIGURE IV.9: Illustration of the 4-steps computation based on the basic pathological example presented in Fig.IV.1: skeletonization & distance transform (A), conversion of space allowance into Social Cost (B) and decreasing wave propagation (C)

IV. 5.3 Skeleton & Space Allowance

Determining the skeleton of a binary occupancy grid and the minimal distance to obstacle cells are well-known problems in the literature, and can respectively be done using skeletonization [148] and distance transform [149] algorithms.

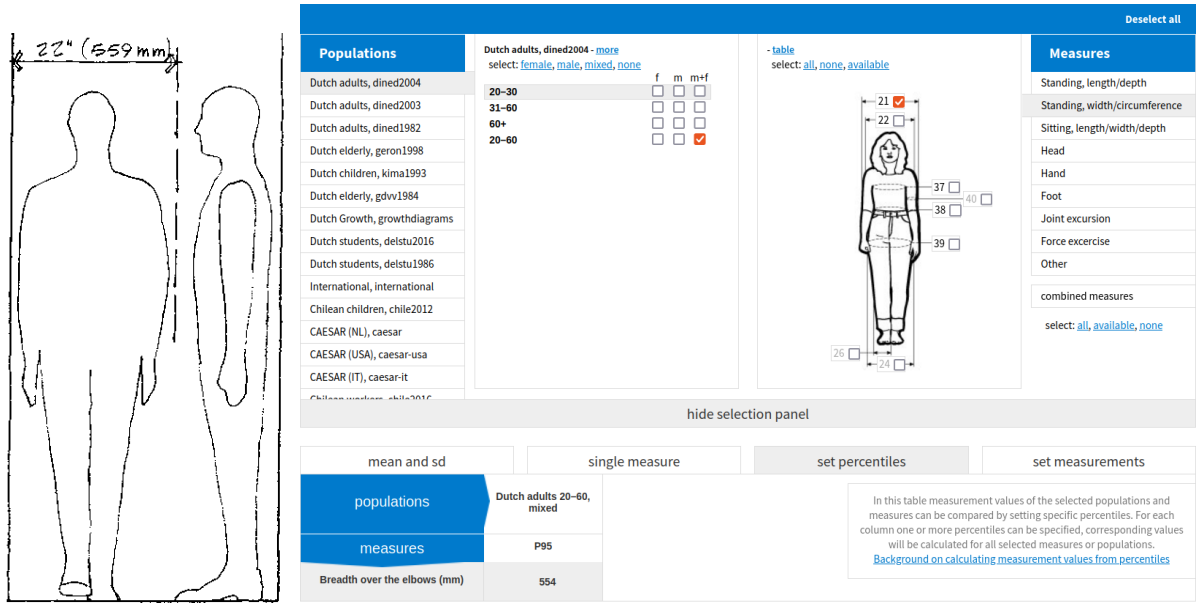
Given the tremendous amount of choice of skeletonization algorithms in the literature, we focused our choice on three of the most popular: Zhang-Suen [150], Guo-Hall [151] and Medial-Axis [152]. We ended up choosing the Guo-Hall algorithm, because, it has a very good contour noise immunity, and does not produce end-of-edge artifacts like Zhang-Suen.

Just like the Medial-Axis method, it guarantees the preservation of components, which is paramount to properly represent passageways. An example of such skeletonization is available in Fig.IV.9A.Top.

To express the original space allowance of passageways (i.e. their "broadness"/"narrowness"), we must measure the minimum **euclidean** distance from fixed obstacles (in contrast to taxicab or chessboard distance), since passageways do not necessarily follow perfect diagonals or straight lines in the grid. We thus use the Euclidean Distance Transform [149] on the binary occupancy grid of fixed obstacles, resulting in a costmap where each cell contains the euclidean distance to the nearest obstacle (in meters), as illustrated in Fig.IV.9A.Bottom. The minimum distance value $d_{min}(x, y)$ in a cell belonging to the skeleton $SK(G(F))$ corresponds to half the space allowance associated with this cell. For example, the left-most cell in the corridor has a minimum distance of 0.45 m from the closest wall, offering a space allowance of 0.9 m (i.e. an entity/group of entities 0.9 m wide could stand in this cell).

IV. 5.4 From Space Allowance to Social Occupation Cost

To transform the previously computed minimum distance $d_{min}(x, y)$ in each skeleton cell $(x, y) \in SK(G(F))$ into a meaningful social cost that expresses the AVOID-NARROW rule, we must use real-world measurements that define what is narrow or wide **for humans**. Basically, we want the cost to be maximal for any space allowance that is below a reference human diameter d_h, that is, any minimum distance value $d_{min}(x, y)$ below a reference human radius $r_h = \frac{d_h}{2}$, then decrease as the space allowance increases.

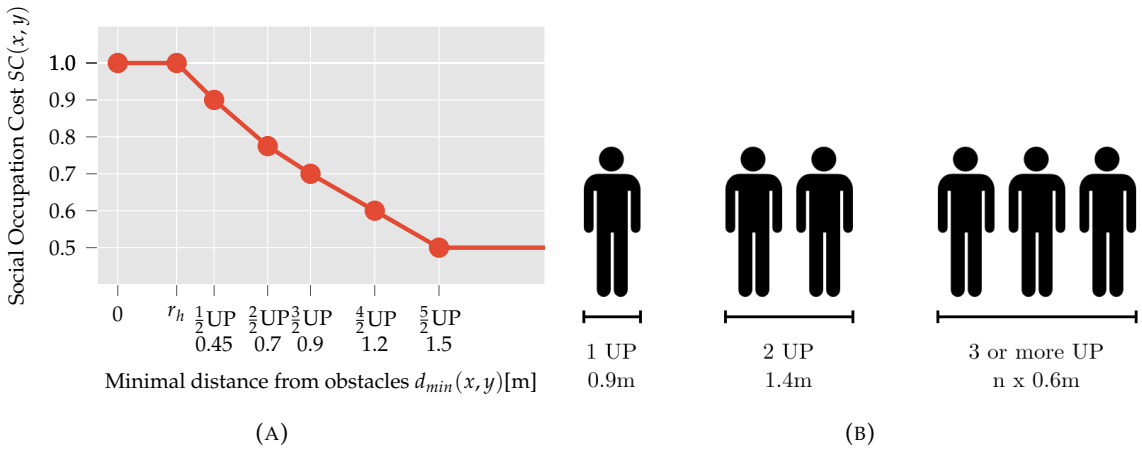


(A) (B)

FIGURE IV.10: Reference human diameter d_h in the literature. (A) Illustration from Mitton and Nystuen's book on interior design we use as reference in this chapter [142] showing their "average adult space clearance for ambulatory human movement". (B) Screenshot of the DINED database interface showing our measurement.

To find a convincing d_h value, we turn to anthropometrics, and use the DINED database, the largest freely accessible resource of human body measurements [153]. We derive the value of d_h from standing breadth over the elbows, which is the widest measurement in a standing posture with arms at rest (Cf. Fig.IV.10b). According to the most recent 2004 data for Dutch adults between the age of 20 to 60, we can reasonably cover about 95% of the population by setting $d_h = 0.55m$ (i.e. $r_h = 0.275m$). This value is coherent with the average adult space clearance for ambulatory human movement of 0.559 m provided by Mitton and Nystuen [142] (Cf. Fig.IV.10a).

Then, to determine a relevant decrease profile, we turn to a commonly used law-defined measurement in French construction [147]: "UP" or "Unités de Passage" (litt. "Passage Units"). UPs describe the minimal space allowance between obstacles/walls for the passageway to allow proper emergency evacuation for n persons: $1\text{UP} = 0.90m$ (meant to comply with standard wheelchair width), $2\text{UP} = 1.40m$, and for three and beyond $n\text{UP} = n * 0.60m$ (Cf. Fig.IV.11b). An indoor passageway is rarely more than $5\text{UP} = 3m$ large because the same law recommends increasing the number of passageways rather than create ones that are more than 5UPs large: therefore, we choose to keep the social cost constant beyond 5UP.



(A) (B)

FIGURE IV.11: (A) Social occupation cost as a function of the minimal distance from obstacles: $\forall(x,y) \in SK(G(F)), SC(x,y) = f_{conv}(d_{min}(x,y))$. (B) Illustration of the concept of "Unités de Passage" (litt. "Passage Units").

For the sake of keeping social occupation cost values comparable, we reduce the interval of acceptable values to $[0,1]$. The cost values associated with each UP between 1 and 5 can be adjusted depending on the relative importance users give to preserving space allowance. Fig.IV.11a shows a set of values we have found to yield a reasonably convincing social occupation cost representation in our experiments discussed in Section IV. 7. Intermediate values are determined by simple linear interpolation, resulting in the social cost conversion function in Equation IV.3. Again, the most relevant properties of this function are its decreasing nature and plateaux; beyond this, one may simplify or complexify the curve as they see fit for their own purposes.

$$\forall (x,y) \in SK(G(F))$$
$$f_{conv} : \mathbb{R}^+ \to \mathbb{R}[0,1]$$
$$f_{conv}(x,y) = \begin{cases} 1 & \text{if } d_{min}(x,y) \in [0, r_h[\\ \frac{-4}{7} \cdot d_{min}(x,y) + \frac{81}{70} & \text{if } d_{min}(x,y) \in [r_h, \frac{1}{2}\text{UP}[\\ -0.48 \cdot d_{min}(x,y) + 1.116 & \text{if } d_{min}(x,y) \in [\frac{1}{2}\text{UP}, \frac{2}{2}\text{UP}[\\ -0.4 \cdot d_{min}(x,y) + 1.06 & \text{if } d_{min}(x,y) \in [\frac{2}{2}\text{UP}, \frac{3}{2}\text{UP}[\\ \frac{-1}{3} \cdot d_{min}(x,y) + 1 & \text{if } d_{min}(x,y) \in [\frac{3}{2}\text{UP}, \frac{4}{2}\text{UP}[\\ \frac{-1}{3} \cdot d_{min}(x,y) + 1 & \text{if } d_{min}(x,y) \in [\frac{4}{2}\text{UP}, \frac{5}{2}\text{UP}[\\ 0.5 & \text{if } d_{min}(x,y) \in [\frac{5}{2}\text{UP}, +\infty[\end{cases} \tag{IV.3}$$

IV. 5.5 Propagation

As explained in the introduction of Section IV. 5, we need a propagation procedure that guarantees a decreasing cost from the skeleton values, in order to guarantee the benefit of getting the obstacle away from the "middle of space".

For this, we use a variant of the Wavefront Propagation algorithm [154] where we start from the skeleton's set of cells SK. We then iteratively mark all unlabeled 4-neighbors cells with the λ-decayed minimum value of their own neighbor cells from the previous iteration, with $\lambda \in]0,1[$, as described in equation IV.4. Since the choice of λ value depends on the grid resolution, we will discuss it in Section IV. 7.

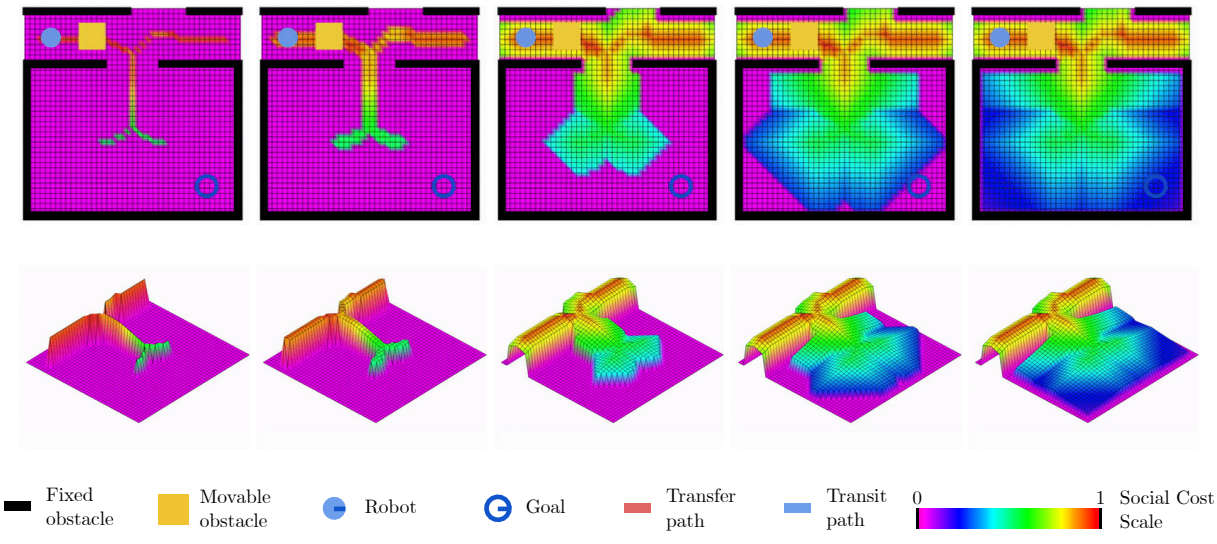


FIGURE IV.12: Illustration of the wave-propagation procedure of the social occupation cost based on the basic pathological case introduced in Fig.IV.1. First row is top view, second row is a 3-dimensional view of the costmap - the robot, movable obstacle and goal are only shown for the sake of readability, but do not affect the social occupation costmap computation.

$$f_{prop}: W \rightarrow \mathbb{R}[0,1]$$
$$f_{prop}(x,y) = \begin{cases} f_{conv}(x,y) & \text{if } (x,y) \in SK \\ \lambda \cdot min(Neighbors(x,y)) & \text{if } (x,y) \notin SK \end{cases} \quad \text{(IV.4)}$$

The process and resulting costmap are illustrated in Figure IV.12. As expected, the middle of space is populated by ridges created by local social occupation cost maxima, and the narrowest spaces are associated with the highest costs, encouraging placement choices outside the corridor and away from the room's center, especially in front of the opening.

IV. 6 Integrating the social occupation cost to NAMO Algorithms

IV. 6.1 Relevant algorithms for this integration

As stated at the end of our problem definition in Section IV. 4, **minimizing both the displacement cost of existing NAMO algorithms and our social occupation cost is a typical multi-objective optimization problem**. Such a problem is generally solved using the concept of *Pareto optimality*, and more precisely by finding the *Pareto front*. By definition, finding the Pareto Front of our problem would require computing the set of all valid plans P_{pareto} for which there is no action to be added that could decrease the robot displacement cost $C(\mathcal{P})$, without increasing the total social occupation cost $SC(\mathcal{P})$. Then, we would need to select one plan to execute among the ones in the Pareto front P_{pareto}: to do so, we would have to favor either the robot displacement cost $C(\mathcal{P})$ or the total social occupation cost $SC(\mathcal{P})$.

As the above definition suggests, **computing the Pareto front P_{pareto} would require the computation of all valid NAMO plans - although**, the literature we have summarized in our NAMO state of the art in Section II. 1 clearly states that **such an exploration is unrealistic for all NAMO problems**. Even when considering only a subset of NAMO problems, as (Stilman, 2005) and (Wu&Levihn, 2014)'s algorithms do, **systematically computing the corresponding subset of all valid NAMO plans is still difficult, and should only ever happen when the problem is in itself difficult** for the algorithms - certainly not every time the algorithm is called.

In the case of (Wu&Levihn, 2014)'s algorithm, trying to replace the displacement cost optimality by a Pareto optimality would require dropping two search space reduction strategies: the entire obstacle selection routine (Cf. Section III. 2.1.1 and Alg. 3) and the obstacle manipulation planning bound (Cf. Section III. 2.1.1 and Alg. 4, l.24). As these two strategies rely on the property that individual action's displacement cost positively accumulate, they would simply not apply in this multi-objective optimization setting. Both cited functions would need to be rewritten so that all obstacles would be evaluated, and all valid plans would be computed for each obstacle. The problem is that, according to Levihn [34], these two strategies are each

individually responsible for an 80 to ∼98% reduction in computational requirements of the algorithm. **Thus, even if it is possible to adapt (Wu&Levihn, 2014)'s algorithm while keeping its optimality property, it would simply cripple the algorithm's computational efficiency**.

On the other hand, (Stilman, 2005)'s algorithm already surrendered optimality, to rather guarantee completeness for solving L_1 problems in reasonable time [3], which they achieved through their obstacle choice heuristic (Cf. Section III. 2.2.1 and Alg. 8). If there is no need to maintain an optimality proof, then there is no need to compute all valid plans, to find the Pareto front and picking "the" best plan. Rather, **we can focus on formulating a useful heuristic to find a *good compromise***. This heuristic will thus be applicable to any other NAMO algorithm that does not require an optimality proof, which is the case for most of the literature, according to our NAMO literature survey in Section II. 1. This is why we shall base all our experiments until the end of this thesis upon (Stilman, 2005)'s algorithm.

IV. 6.2 A heuristic compromise cost

We thus propose to compute and use a heuristic compromise cost $CC(x, y)$ in the transfer planner (syn. obstacle manipulation planner) of NAMO algorithms, for the set of potentially reachable cells to move the selected obstacle at. We do that with the following process, illustrated in Fig.IV.13:

1. Temporarily remove the selected obstacle M_i from the robot's binary occupancy grid.
2. Inflate the grid by the inscribed circle radius of M_i. This operation allows to find a subset of potentially acceptable cells for M_i.
3. Compute a goal-less Dijkstra search from M_i's center cell. The set of potentially reachable cells for moving M_i is thus obtained, with an underestimate of the distance $d_{M_i}(x, y)$ to reach them[5].
4. Compute the compromise cost CC for each potentially reachable cell with a weighted sum of the following normalized costs: occupation cost ($SC(x, y)$), distance $d_{M_i}(x, y)$ and a heuristic euclidean distance to NAMO goal $d_{goal}(x, y)$:

$$CC(x, y) = \frac{w_{M_i} * d\prime_{M_i} + w_{goal} * d\prime_{goal} + w_{SC} * SC\prime}{w_{M_i} + w_{goal} + w_{SC}} \tag{IV.5}$$

   where $SC\prime$, $d\prime_{M_i}$, and $d\prime_{goal}$ are respectively normalized versions of SC, d_{M_i}, and d_{goal} in $[0, 1]$.

The obtained Comprise Cost grid can then be used in the transfer planner of NAMO algorithms in two non-exclusive ways:

1. it can be used to **bound the action space exploration**, modifying the search exit condition, so the search is stopped early when a solution leaves the evaluated obstacle's center in

[5]The goal cells are filtered out of this set afterwards so that the robot can't put the obstacle on it.

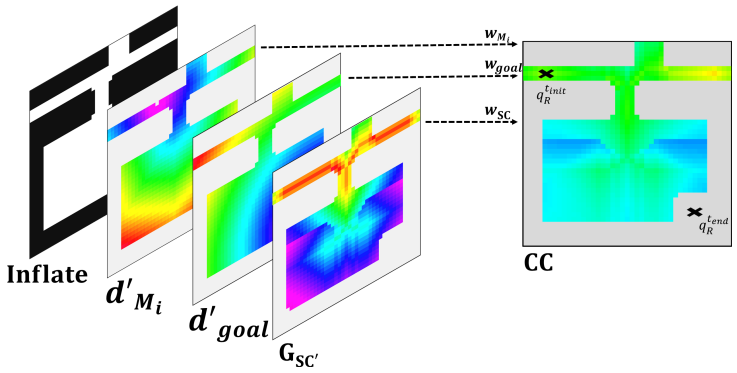


FIGURE IV.13: Illustration of the Compromise cost (CC) computation steps based on Fig.IV.1

one of the cells within the top n percent best CC cost. This is similar to how (Wu&Levihn, 2014)'s algorithm bounded their search with a cost estimate of the currently evaluated obstacle manipulation plan (Alg. 4, l.24), albeit without guaranteeing optimality, as previously discussed.

2. before starting the search, it can be used to select the obstacle cell with the least combined cost as target cell, and focus the search towards it. Even if the cell may not be reachable, the previously discussed bound allows exit if a cell with a close-enough compromise cost is reached.

These two modifications preserve the algorithm's completeness, as they do not prevent the search to explore all reachable transfer configurations if none of the top n percent best CC cost cells can be reached. Thus, any problem that can be solved by the baseline algorithm is guaranteed to be solvable by the modified social version - which is significant to make a fair comparison between the two.

We applied both modifications to our (Stilman, 2005) improved algorithm presented in Section III. 2.2.2, using an A^* search instead of the Dijkstra search to focus on the best compromise cell for the S-NAMO variant[6] (Cf. Alg.9, l.2).

IV. 7 Experiments

Now that we have defined a social cost model that does not demand complex semantic knowledge, found a way to combine it with existing NAMO Algorithms' transfer planners, and chosen a relevant NAMO algorithm as a baseline to compare against, let us experimentally explore its effects. All experiments were run on an Intel Core i7-8850H CPU @2.60GHz with 16 GB of RAM.

[6]The accompanying video shows how both algorithms unfold:
https://team.inria.fr/chroma/files/2020/07/iros_2020_Renault.mp4

IV. 7.1 Simulation parameters

Before going any further, let us first define the algorithm's parameters values for the experiments: these are summarized in Table IV.1. The choice of values has been manually tuned across the four different experiment environments illustrated in Fig.IV.15:

- The grid resolution of 0.1 m we have adopted is a common choice in the NAMO literature, as it can represent details of many objects in human environments while not needlessly multiplying the number of cells and thus computational requirements;
- The decay factor λ is mainly tied to the grid resolution choice, and has been decided upon so that the decrease profile was not too steep in the 4 environments. The chosen value of 0.97 can reasonably be expected to generalize well with other human indoor environments, supposing the same grid resolution. The finer the resolution, the lower the decay factor value should be to get a similar social costmap;
- For the Compromise Cost weights w_{M_i}, w_{goal} and w_{SC}, we started out with a 50/50 distribution between social and distance-related weights (that is, $w_{M_i} + w_{goal} = w_{SC}$). We quickly favored the obstacle movement distance estimate over the distance-to-goal estimate (more precisely, $w_{M_i} = 10$ and $w_{goal} = 2$), as obstacle transfers are inherently costlier than navigation. In the end, we chose to slightly favor social cost further ($w_{SC} = 15 > w_{M_i} + w_{goal}$), as we wanted our algorithm to slightly favor socially-relevant choices over displacement-minimizing ones.
- The compromise cost search bound n is set to 1% of the selected best compromise cost. We have found this to be a good compromise between bringing the obstacle close enough to the selected cell, while not wasting further computational time trying to reach the one specific chosen cell.

Square grid resolution for G_{SC}	0.1 m
Decay factor for social cost propagation λ	0.97
Compromise cost weight for obstacle movement distance estimate w_{M_i}	10
Compromise cost weight for distance-to-goal estimate w_{goal}	2
Compromise cost weight for social cost estimate w_{SC}	15
Compromise cost search bound n	1%

TABLE IV.1: S-NAMO algorithm parameters values for the experiments

IV. 7.2 Evaluation criteria

We now need criteria for measuring both the displacement cost efficiency and social acceptability of solutions. For displacement cost efficiency, as in all of NAMO literature, we use the total euclidean distance for transfer and transit paths, respectively $L_{transit}$ and $L_{transfer}$ ($L_{transit} + L_{transfer} = L_{total}$) - rather than using the algorithm's specific displacement cost definition.

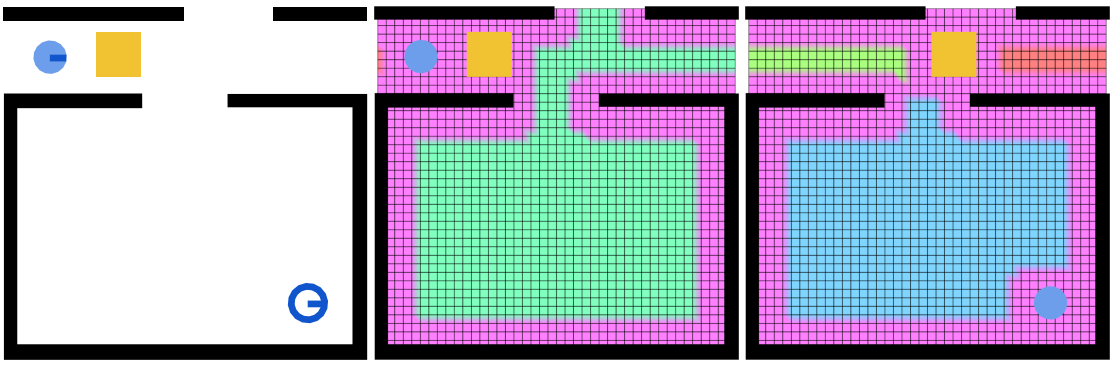
As for social acceptability, we define the following criteria:

- **Total Social Occupation Cost** ST : Derived from our model, it is the sum of SC for all movable obstacles at time t. It represents the global space accessibility disturbance for humans within a single criterion.

$$ST(W^t) = \sum_{\forall M_i \in M} SC(M_i^t) \tag{IV.6}$$

- **Number of components** Nc : In the grid representation of W^t inflated by the reference human radius r_h defined in Section IV. 5, it is the number of interconnected sets of cells $C_h^i(W^t)$ that make up the global set of accessible cells for humans $C_h^{acc}(W^t)$, as illustrated in Fig.IV.14. Free space components are as much of an essential topological characteristic of space for humans, as they are for robots: disconnecting (i.e. adding) space components for humans can arguably be considered the worst navigability/accessibility affordance violation - hence why it is essential to measure it.

$$Nc(W^t) = |\{C_h^i(W^t)\}| \tag{IV.7}$$



(A) Initial state of our basic scenario of Fig.IV.1a

(B) Initial two free space components *for humans* (two-cells small red component left to the robot)

(C) Final state, with three space components *for humans* after the NAMO resolution of Fig.IV.1b

FIGURE IV.14: Grid representation of W^t inflated by the reference human radius r_h. Pink cells are inaccessible cells *for humans*, other cell colors each correspond to a separate free space component *for humans*.

- **Space fragmentation percentage** $frag$: In the same grid as before, it relates the size of the biggest free space component $C_h^{max}(W^t)$ to the total size of free space $C_h^{acc}(W^t)$. It allows to measure the accessible proportion of space for a human if it were present in the biggest free space component, relativizing the previous criterion of number of components. For example, if an environment were only 5% fragmented, but had dozens of very small free space components, it would arguably remain very much navigable for a human without having to resort to manipulation. Conversely, if the same environment were to be split into three equal components, resulting in a $\sim$ 33% fragmentation, the likelihood of the human having to resort to manipulation to navigate the environment between any two

points would conceivably be higher.

$$frag(W^t) = 1 - \frac{|C_h^{max}(W^t)|}{|C_h^{acc}(W^t)|} \tag{IV.8}$$

A plan improves social acceptability by lowering any of these criteria between $W^{t_{start}}$ and $W^{t_{end}}$, t_{end} being the time at which the robot has executed all its actions. Finally, we also provide the total planning time $T_{planning}$ to represent the computational cost of each plan; this value is averaged (and provided with standard deviation), as it may vary across multiple runs of a same experiment.

IV. 7.3 Results

IV. 7.3.1 Short-term experiments

We first evaluated our approach with four short-term scenarios of increasing complexity, each shown in a line of Fig.IV.15. "Short-term" refers to the low number of successively executed goals in these scenarios (mainly 1 per scenario), comparatively to our second set of experiments in the following Section IV. 7.3.2 (100 per scenario). These "short" and "long"-term designations are thus not used as strictly opposite terms, but designate a spectrum, where one-goal scenarios are however definitely considered short-term. Since, the current NAMO literature only covers short-term, one-goal scenarios, it makes sense to first study such scenarios.

In Fig.IV.15, scenario [A] represents the same corridor&room scenario we have been studying since the beginning of this chapter (Cf. Fig.IV.1). Scenario [B] showcases 4 rooms joined by corridors, each containing a goal, which are executed counter-clockwise starting from the right (for the sake of readability, only the state after the first goal is shown in the third and fourth columns). Scenario [C] exhibits a room filled with unmovable tables (black squares) and miscellaneous movable obstacles "after a party". Finally, scenario [D] is a real-world representation of our lab's second floor, obtained from a 3D scan - a first goal lies in the second room from the left at the top, while a second goal is in the first room from the right at the bottom. **It must be reminded for proper understanding that the free space components represented in the last three columns are computed using the reference human radius r_h, and not the (smaller) robot radius**. Thus, even if the robot successfully connected two components to pass, the components may still be disconnected for humans, as can be seen in the 6th column for scenarios [A], [B] and [D].

Table IV.2 compares the values of the previously presented evaluation criteria (Cf. Section IV. 7.2), obtained in these 4 scenarios between the baseline (Stilman, 2005) NAMO algorithm and its S-NAMO counterpart. Since scenarios [B] and [D] consist of multiple goals, displacement and computation costs for are cumulated over goals. Each scenario is run 50 times in order to obtain a meaningful average of planning time $T_{planning}$. We further comment these results hereafter.

	$L_{transit} + L_{transfer}$ [m] + [m]	$ST(W^{t_{start}}) \rightarrow ST(W^{t_{end}})$	$Nc(W^{t_{start}}) \rightarrow Nc(W^{t_{end}})$	$frag(W^{t_{start}}) \rightarrow frag(W^{t_{end}})$ [% → %]	$T_{planning} \pm \sigma$ [s]
[A]					
Baseline	4. + 1.1	46.8 → 43.0	2 → 3	0.4 → 12.6	0.5 ± 0.2
S-NAMO	2.5 + 2.8	46.8 → **15.0**	2 → **1**	0.4 → **0**	1.1 ± 0.2
[B]					
Baseline	33.0 + 1.2	61.0 → 44.9	3 → 4	23.6 → 74.3	13.8 ± 1.6
S-NAMO	35.7 + 3.4	61.0 → **23.7**	3 → **1**	23.6 → **0.**	**2.0** ± 0.3
[C]					
Baseline	11.2 + 0.6	559.0 → 556.5	4 → 3	45.5 → 28.5	1.3 ± 0.2
S-NAMO	13.0 + 2.5	559.0 → **545.9**	4 → 3	45.5 → 28.6	3.1 ± 0.4
[D]					
Baseline	31.8 + 1.0	105.4 → 116.7	14 → 15	47.2 → 45.5	28.1 ± 1.8
S-NAMO	35.4 + 4.5	105.4 → **79.3**	14 → **13**	47.2 → **5.0**	26.5 ± 1.7

TABLE IV.2: Short-term single-robot experiments performance criteria comparison table. As a reminder: displacement and computation costs for [B] and [D] are cumulated over goals, ST and Nc have no units and $T_{planning}$ is given as average with standard deviation σ over 50 runs.

Scenario [A] shows the same typical pathological case scenario we have studied along this chapter, where (Stilman, 2005)'s algorithm, or any of the other current NAMO algorithms, would cause unintentional connectivity loss - as we have already stated in Section IV. 1. At the cost of little extra computation (although twice as long, it only adds 0.6 seconds in absolute) and further displacement of the obstacle (the total traversed distance is only 4% greater), the S-NAMO approach results in complete defragmentation - when baseline NAMO further fragments the environment in the eyes of our reference human of radius r_h, as can be seen in Fig.IV.15's 6th column. Unsurprisingly, since our social approach considers obstacle placements other than the nearest ones that open its way, the transfers get longer and costlier.

Scenario [B] similarly tends to show that our social approach is - at least in the short term - costlier in terms of displacement cost: here the total traversed distance is 14% greater for S-NAMO compared to the baseline, and the transfer distance is almost twice as big (although, only 2.2 m longer in absolute). But more importantly, this scenario also allows to highlight an interesting phenomenon: NAMO algorithms, when faced with a movable obstacle blocking a tight intersection branch they need to pass, will naturally tend to block another branch because of the sole focus on displacement cost, as illustrated in Fig.IV.15's 3rd column. In baseline NAMO, as the robot gets from one room to the next, it repeatedly blocks another corridor and systematically induces an additional computation cost, that after 4 goals is already almost 7 times greater than our S-NAMO approach - because computing a NAMO plan is costlier than a pure navigation plan. Our S-NAMO approach quickly evacuates the obstacle from the intersection into the left room on the first goal, greatly facilitating any future navigation.

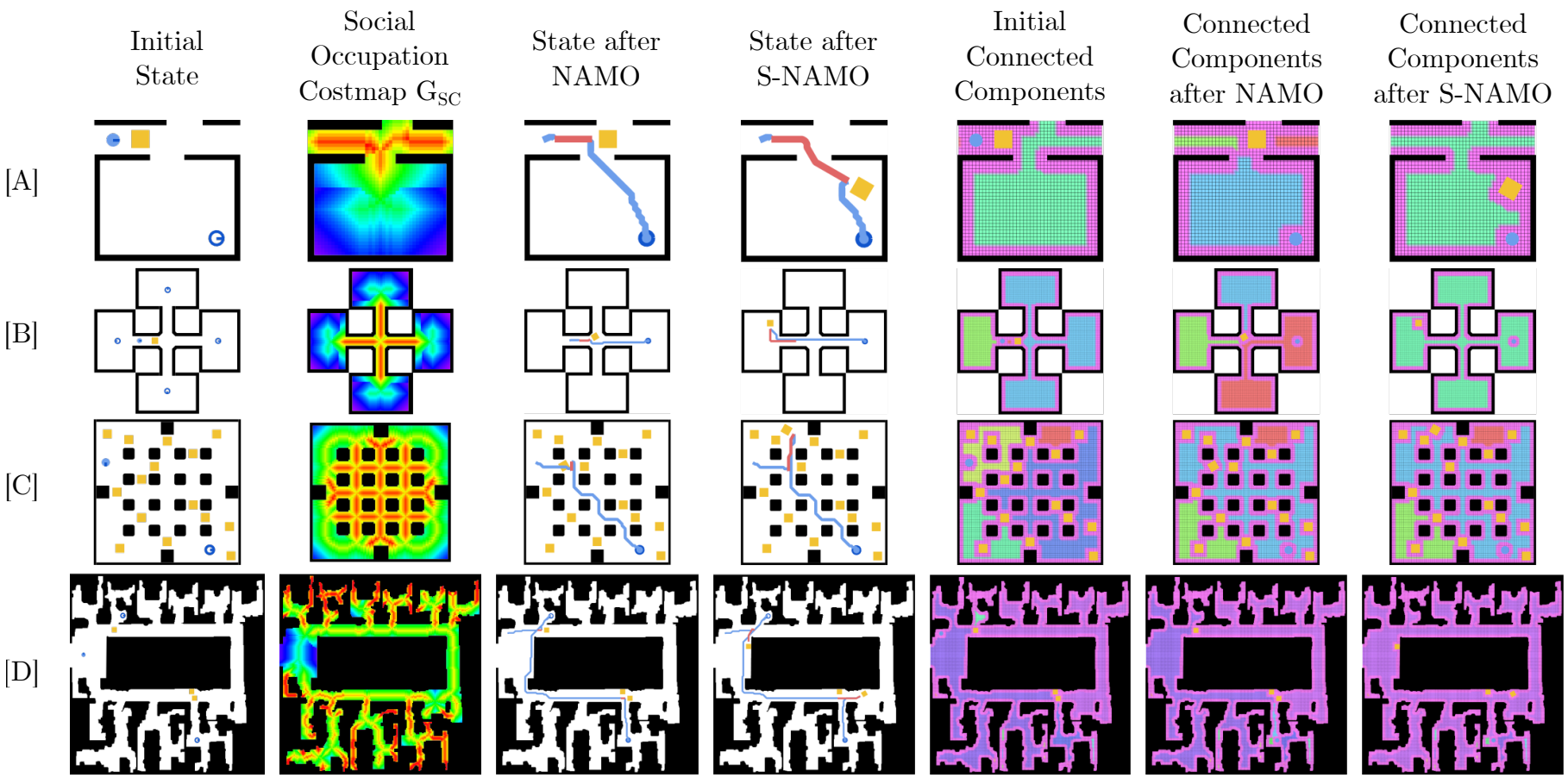


FIGURE IV.15: Experimental scenarios & NAMO vs. S-NAMO results. [A] "Corridor And Room" (Fig.IV.1), [B] "Crossing", [C] "Intersections" and [D] Real-world map of our "CITI Laboratory". In [A] and [C], the robot is only required to reach one goal, while in [B] and [D], it has to sequentially reach respectively 4 (counter-clockwise, starting from the right room in [B]) and 2 goals (top-left then bottom-right rooms in [D]). In the 3rd and 4th columns, only the final state after these multiple goals is shown.

In scenario [C], the same phenomenon observed in [B] occurs: the NAMO approach blocks another intersection branch while the S-NAMO approach drives the obstacle to the environment's periphery. This reinforces the evidence that a better obstacle placement from a social point of view comes at the price of an increase in traversed distance - here, a 31% overall increase, with a 4 times longer transfer distance. But more importantly, this scenario shows that, out of the three criteria we propose to evaluate social acceptability, only the one based on our social cost computation ST allows us to differentiate the quality of solutions that produce almost the same change in number of components ($4 \rightarrow 3$) and space fragmentation ($\sim$45% $\rightarrow$ $\sim$28%).

Finally, Scenario [D] shows that our approach is capable of growing up to scale in big realistic environments. Indeed, despite the environment size and complex topology, corridors around the middle obstacle and tight passages between desks in the surroundings, the offices' passageways are properly detected as less appropriate for obstacle placement than the wider areas like office corners, the entry hall on the left and the big intersection on the right. The results in Table IV.2 show that despite this increase in complexity compared to the other scenarios, our approach still succeeds in achieving more socially acceptable solutions than the baseline approach[7], *for a similar computational cost*.

Preliminary Conclusion: This last scenario, together with the results of scenario [B], go to show that while our approach may incur higher displacement costs (which was to be expected), *it does not necessarily increase computational costs*. These experiment tend to show that using our social cost model seems to systematically yield better results social-wise, not only from the perspective of our own model-dependent criterion (ST), but also with independent topology and geometry-based criteria (Nc and $frag$).

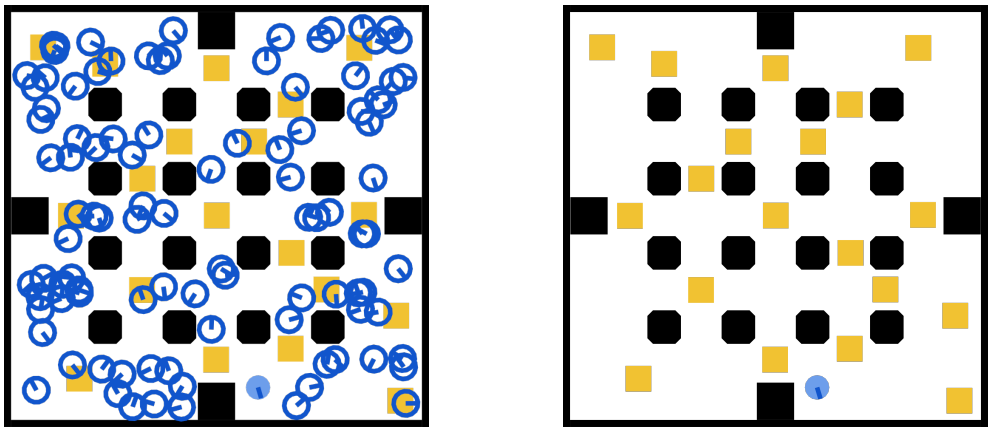
IV. 7.3.2 Long-term experiments

We observed in our previous experiments, notably in the "Crossing" [B] scenario, that our social cost model could not only achieve better social adequacy, but could also positively affect other evaluation criteria such as computation time in longer-term experiments. In scenario [B], if the robot were asked to repeat several times the four-goal sequence of the scenario, at some point, the NAMO baseline would inevitably become costlier in terms of traversed distance than the S-NAMO alternative, because of the repetitive additional transfer actions between the intersection's branches. Rather than studying further this simple scenario, let us explore whether S-NAMO can positively affect non-social evaluation criteria in the more complex environment of "Intersections" [C].

To do so, we executed 200 experiments in this environment, randomizing the start robot configuration and a sequence of 100 goals each time, but not randomizing the position of movable obstacles, since they need to be put in relevant places (keyholes) to create actual NAMO

[7] The high amount of detected components is due to the chosen reference human radius r_h: in the offices, space is considered fragmented between some desks because of this. This does however not make the comparison any less relevant.

problems. We again compare the NAMO baseline with our S-NAMO approach: an example of such an experiment is shown in Fig.IV.16. In order to compare the baseline NAMO with the S-NAMO approach, we compute the average and standard deviation of each evaluation criterion over the 200 experiments, and study their evolution over time in the graphs shown in Fig.IV.17 - which we discuss hereafter.



(A) Initial configuration with the 100 goal-sequence

(B) Initial configuration without the goals (for the sake of readability)

(C) After 100 random goals with NAMO

(D) After the same goals with S-NAMO

FIGURE IV.16: Illustration of one of the 200 randomized experiments with a single robot in the "Intersections" environment.

The first conclusion is that S-NAMO still significantly improves the environment's state, making it significantly clearer and tidier. This is qualitatively supported by the observable difference between the environments yielded by NAMO and S-NAMO, in Fig.IV.16c and IV.16d respectively. While NAMO fundamentally connects the environment (as shown by the fast reduction in number of components in Fig.IV.17b), S-NAMO goes a step further and "tidies" it up, eventually almost fully defragmenting it, as shown by Fig.IV.17c. By the end, the average fragmentation reaches near 0% with a standard deviation below 1% for S-NAMO, while it converges around 5% with a standard deviation of about 10% for NAMO. Since S-NAMO is encouraged to leave moved obstacles closer to static obstacles, it also increases the number of human-traversable free cells by almost 8% (Cf. Fig.IV.17d), while decreasing the total social cost

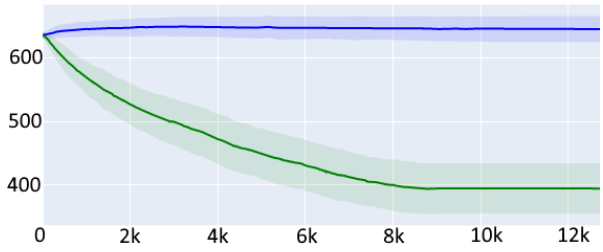
by more than 40%, according to Fig.IV.17a.

The second conclusion derives from the first one: a more accessible space for humans is also a more accessible space for the robot[8]. Improvements to the environment's state are also reflected by a reduction of the total path length (Cf. Fig.IV.17h, IV.17e) leading to faster goal completion than baseline NAMO. Indeed, as shown in Fig.IV.17g, after around 20 goals, S-NAMO starts to visibly allow the robot to accomplish its goals faster in terms of simulated time steps (i.e. number of executed actions), reaching a plateau about 25% steps sooner at around 90 successful goals on average. As NAMO eventually reaches its plateau of successful goals, its accumulated traversed distance is more than 20% than that of S-NAMO. The number of transfer operations (the number of continuous *Manipulate* operator sequences) is also reduced by around 25%, which is a net positive, since manipulations present an additional failure risk in real-world applications. The reason why both NAMO and S-NAMO however do not systematically succeed in completing 100% of their goals is due to resolution failure of the underlying improved (Stilman, 2005) algorithm, that cannot solve NAMO problems where more than one obstacle cover the goal configuration.

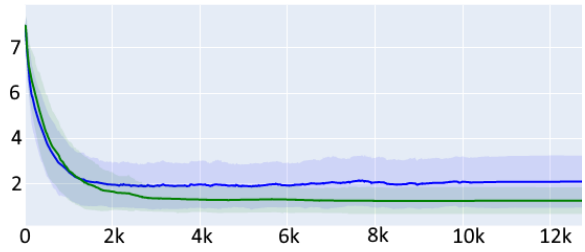
It is to be reminded that (Stilman, 2005)'s algorithm is not optimal, and more exactly, it simply won't try to compute a plan including obstacle manipulations if it can reach its goal with a pure navigation plan. That is why the curvature of the number of transfers graph (Cf. Fig.IV.17h) becomes less steep as soon as most components have been connected (Cf. Fig.IV.17b), after about a dozen goals on average. The reason why transfers still occur after that is because some goals are still covered by obstacles, and trigger a transfer plan computation. This also explains why the computation time of S-NAMO remains consistently higher (Cf. IV.17f) compared to NAMO in this scenario.

[8]Here, this is particularly true for robots smaller than humans due to our use of a human-size reference radius r_h; if robots were larger than humans, then we could simply adjust the reference radius and skeleton conversion function f_{conv} to be coherent with the robot's size, resulting in a similar conclusion.

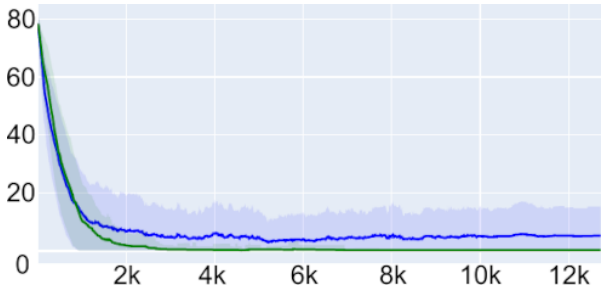
FIGURE IV.17: Averaged evaluation criteria graphs (with standard deviation) of the 200 randomized experiments in the "Intersections" environment presented in Fig.IV.16, baseline NAMO in blue, S-NAMO in green. **Ordinate (y-axis)**: values for the criterion given in caption. **Abscissa (x-axis)**: number of elapsed simulation time steps.



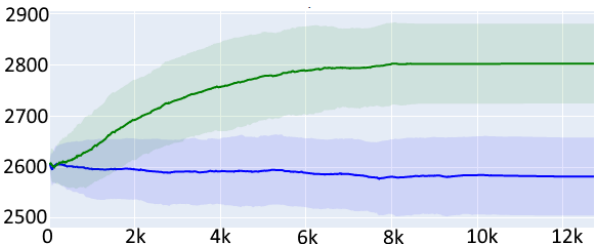
(A) Total Social Occupation Cost $ST(W^t)$ [UA]



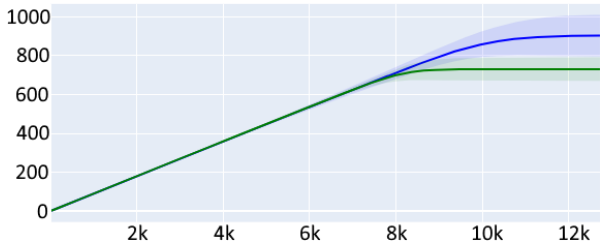
(B) Number of free space components $Nc(W^t)$



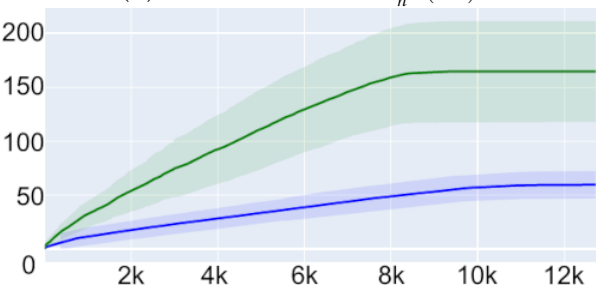
(C) Space fragmentation $frag(W^t)$ [%]



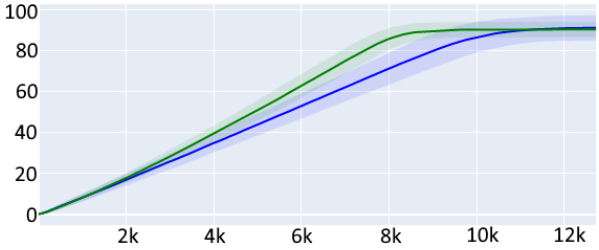
(D) Number of free cells $C_h^{acc}(W^t)$



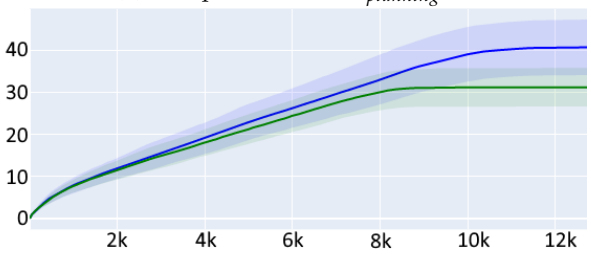
(E) Total robot traversed distance L_{total} [m]



(F) Computation time $T_{planning}$ [s]



(G) Number of successful goals



(H) Number of obstacle transfers

IV. 8 Conclusions

This chapter first introduced the **new problem of General Socially-Aware Navigation Among Movable Obstacles**. Inspired by our Social Navigation literature exploration (Cf. Section II. 2.1.1), we proposed **a first naive social constraint of "taboo zones"**, or forbidden affordance spaces, to express this problem. While this simple approach does not affect optimality of algorithms, it does not scale nor generalize well to any NAMO scenario. After consulting the literature on Affordance Spaces, we realized that *a new model was needed that did not require extensive semantic data and could cover the entirety of the environment*. Hence, **we introduced a new social occupation cost model** representing an affordance space of *accessibility/navigability* based on two heuristic hypotheses: avoiding narrow spaces, and not leaving objects in the middle of space. **This model relies solely on the analysis of the environment's binary occupancy grid of fixed obstacles**. Then, we showed how to extend NAMO to S-NAMO algorithms that result in plans with better social acceptability. **To the best of our knowledge, this is the first approach to propose an answer to social concerns in NAMO problems**.

Experimental results on scenarios of increasing complexity have shown the scaling and generalization capabilities of our method. The same results tend to show that in the short-term, improving social adequacy of obstacle placement comes at the cost of increased robot movement, but that in the longer term, the displacement cost may become lower especially in initially cluttered environments, by tidying them up. **It is to be noted that, to the best of our knowledge, we are the first to explore long-term NAMO experiments**. In our experiments, the computation time of socially-aware obstacle transfers was systematically higher than that of non-aware ones, mainly due to the cost of exploring more robot-obstacle configurations. We however attribute this mainly to the choice of A^* as transfer planning algorithm, which does not focus the search too much for the sake of completeness and optimality. In future experiments, algorithms such as Jump Point Search [155] or RRT* [68] could be used to focus this search more towards the selected best compromise cost cell, likely resulting in reduced transfer planning computation time.

Our model could certainly still be improved, notably in regard to its parametrization. As with any parameterized model of human preference, finding good -or best- parameter values for our social cost model that hold relevance beyond the eye of the experimenter would require feedback from a large and diverse-enough group of humans - which we did not have access to in the context of this work. In the future, it may be relevant to set up a social experiment where humans could rank social costmaps and corresponding S-NAMO robot plans with a larger dataset of indoor environments.

Finally, our adoption of a costmap-based social constraint model leaves the door open to straightforwardly integrating local object semantics-based affordance space models such as those of Jiang et al.[23], or human observation-based ones such as the one of Limosani et al. [22], that we both mentioned in Section IV. 3. This would also require tuning weighting parameters to integrate the different costs into one, but the resulting model would certainly yield even more meaningful and relevant obstacle choice placements.

Chapter V

Multi-Robot NAMO

In addition to the conclusion of our state of the art in Chapter II that social concerns had never been addressed in the NAMO literature, we also remarked that **actual Multi-Robot NAMO had never been addressed either**: that is, having several robots share the same space, concurrently navigating while moving obstacles out of their way without colliding. In the previous chapter, we developed a social model for robots to make NAMO plans that take into account the potential navigation needs of all agents (in other words, the accessibility affordance for humans and other robots), even if they are not within the environment at the moment of plan execution. **In this chapter, we actually introduce other robots in the environment, and define the Multi-Robot NAMO (abb. MR-NAMO) problem**. We then present an implicit coordination framework for existing NAMO and S-NAMO algorithms, allowing us to explore this new context, as shown in Fig.V.1. This figure illustrates the same basic pathological scenario introduced in Chapter IV, but with an additional robot whose movement gets more or less hindered depending on the first robot's plan choice - motivating our discussion in the present chapter on the relevance of our previously presented Social Cost Model in the presence of actual agents.

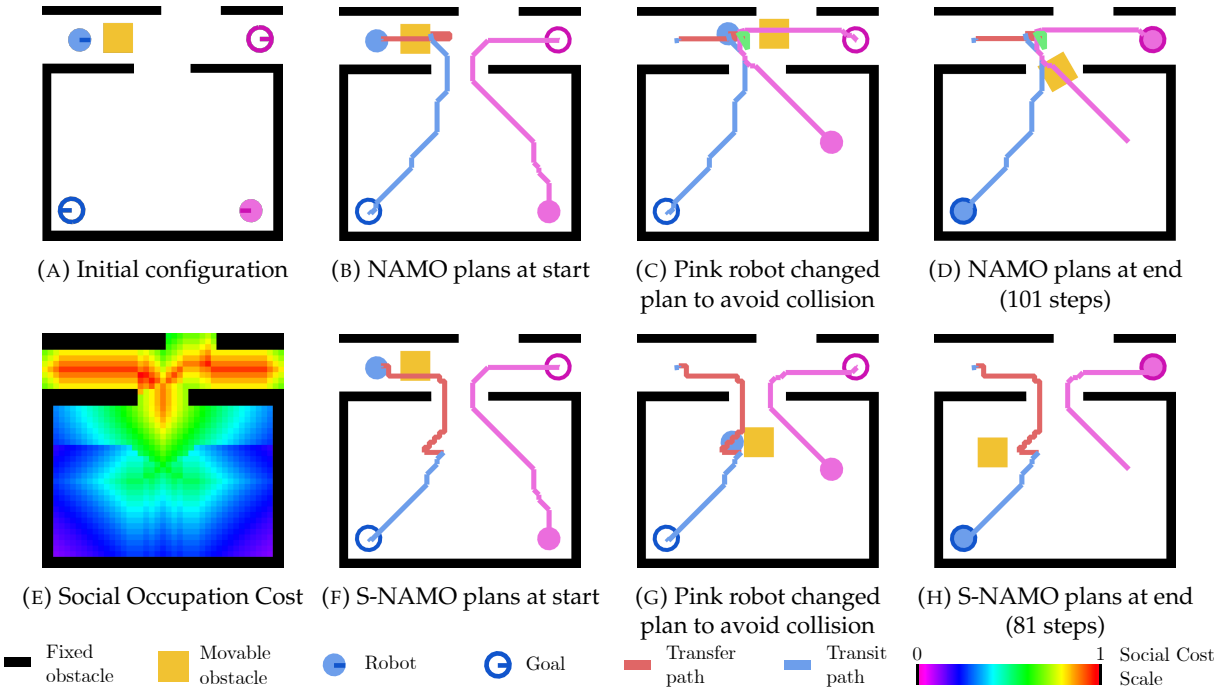


FIGURE V.1: Simple Multi-Robot scenario derived from Fig.IV.1, and resolution by our Implicit Coordination Strategy, without (1st line) and with (2nd line) the Social Cost Model presented in Chapter IV. The blue robot chooses a more appropriate placement for the obstacle using S-NAMO than NAMO, allowing the pink robot to reach its goal without having to move the obstacle again, reducing overall execution time (makespan) by almost 20%.

V. 1 MR-NAMO Problem definition

We build the General MR-NAMO problem in the continuity of the General NAMO problem definition given in Section III. 1. As the name indicates, the purpose of the following section V. 1.1 is to express a definition that is as general as possible; the subsequent section V. 1.2 presents the additional hypotheses we use to constrain the scope of problems covered by this definition as to facilitate implementation and experimentation in the rest of the chapter.

V. 1.1 General MR-NAMO problem

Instead of a single robot, we define $R =< R_1, ..., R_r > \subset E$ as a set of robots (part of the world's entities E). Similarly to how R was previously defined as a separate element from the set of obstacles (syn. objects) O, we keep this separation $R \cap O = \emptyset$: as per this equation, it is hypothesized that a robot cannot *Manipulate* another robot as if it were a manipulable obstacle.

Each robot R_i is assumed to have its own sensing capabilities. As such as we did in Section III. 1.3, we differentiate the *actual world* (syn. *reference world*) W_{ref}, where actions are executed, and the *individual world perception of each robot* W_{R_i}. The notations F_{R_i} and M_{R_i} respectively refer to the sets of obstacles *perceived to be* either fixed or movable by robot R_i.

Each robot R_i has its own action space $\mathcal{A}_{R_i}$, that may or may not include *Manipulate* operators: in the general case we aim to model here, not all robots possess a manipulation capability. This heterogeneity of capabilities means that categorization of obstacles as movable or fixed may not necessarily be the same for each robot. As such, in the reference world W_{ref}, the sets of movable and static obstacles must be written as subsets specific to each robot: F_{ref,R_i} and M_{ref,R_i}. These notations respectively refer to the sets of obstacles *that are actually* fixed or movable for robot R_i.

We explained in Section II. 2.2.1 that a coordination problem's main concern is to prevent **conflicts** - that is, incompatible concurrent access to a same resource. The fundamental conflict type of Multi-Robot coordination problems is of **Space conflict** - in other words, physical collisions. In our formalism, such a conflict occurs if, at a time t within the time of execution of both plans (written as $[\min(P_{R_i}, P_{R_j}), \max(P_{R_i}, P_{R_j})]$), the swept area S of a robot's action $A_{R_i}(t, t+1)$ intersects either with the swept area of another robot's action, or with the space occupied by any obstacle $\in O^t$ that is not manipulated by current said action (written as $M_k \in A_{R_i}(t, t+1)$):

$$\begin{aligned} \exists R_i, R_j \in R, \exists t \in [\min(P_{R_i}, P_{R_j}), \max(P_{R_i}, P_{R_j})], \\ S(A_{R_i}(t, t+1)) \cap S(A_{R_j}(t, t+1)) \neq \emptyset \\ \text{or } S(A_{R_i}(t, t+1)) \cap O^t_{/M_k \in A_{R_i}(t,t+1)} \neq \emptyset \\ \text{or } S(A_{R_j}(t, t+1)) \cap O^t_{/M_k \in A_{R_j}(t,t+1)} \neq \emptyset \end{aligned} \tag{V.1}$$

In a multi-robot setting, multiple robots may manipulate a same obstacle at the same or at a different time, which can result in another type of conflict beyond Space Conflicts, that we call **Movable Obstacle Conflict**. In our formalism, such a conflict occurs if:

- at the same time t, two robots R_i and R_j want to *Manipulate* a same obstacle M_k (at respecting grasps G_n, G_m) and have it follow different expected trajectories $\tau(q^t_{M_k,R_i}, q^{t+1}_{M_k,R_i})$ and $\tau(q^t_{M_k,R_j}, q^{t+1}_{M_k,R_j})$:

$$\begin{aligned} \exists R_i, R_j \in R, \exists t \in [\min(P_{R_i}, P_{R_j}), \max(P_{R_i}, P_{R_j})] \text{ s.t.} \\ A_{R_i}(t, t+1) = Manipulate_{R_i}(W^t, M_k, G_m, \tau(q^t_{R_i}, q^{t+1}_{R_i})) \\ \text{and } A_{R_j}(t, t+1) = Manipulate_{R_j}(W^t, M_k, G_n, \tau(q^t_{R_j}, q^{t+1}_{R_j})) \\ \text{and } \tau(q^t_{M_k,R_i}, q^{t+1}_{M_k,R_i}) \neq \tau(q^t_{M_k,R_j}, q^{t+1}_{M_k,R_j}) \end{aligned} \tag{V.2}$$

- a robot R_i wants to *Manipulate* an obstacle M_k at expected configuration $q^t_{M_k,R_i}$, but another robot's plan would result in the obstacle being in a different configuration $q^t_{M_k}$:

$$\begin{aligned} \exists R_i \in R, \exists t \in P_{R_i} \text{ s.t.} \\ A_{R_i}(t, t+1) = Manipulate_{R_i}(W^t, M_k, G_i, \tau(q^t_{R_i}, q^{t+1}_{R_i})) \\ \text{and } q^t_{M_k} \neq q^t_{M_k,R_i} \end{aligned} \tag{V.3}$$

This being said, and with the inspiration of the MAPF problem definition given in Section II. 2.2.2, we can now formulate our problem:

General MR-NAMO Problem: Given an initial workspace configuration W^{t_0} with a set of robots $R =< R_1, ..., R_r >$, the MR-NAMO problem consists in computing, if they exist, for each robot R_i with an action space $\mathcal{A}_{R_i}$ that may or not allow manipulation of movable obstacles $M_i \in M_{ref,R_i}$, a plan $\mathcal{P}_{R_i}$ from its initial configuration $q^{t_0}_{R_i}$ to a goal configuration $q^g_{R_i}$, that can be concurrently executed while guaranteeing the absence of conflicts between the robots, their manipulated obstacles, and any other obstacles $O_k \in O$.

V. 1.2 Additional hypotheses for our study

In order to explore MR-NAMO for the first time, we need hypotheses that will allow us to sufficiently simplify the problem as to make it possible to experiment within reasonable research time. We ended our state of the art in Section II. 2.2.2 with an argumentation as to how and why we should simplify the General MR-NAMO problem formulated above; let us now formalize this argumentation here:

Homogeneous & Omniscient Multi-Robot System: We shall assume all robots share the same action space $\mathcal{A}$ (that necessarily includes at least one *Manipulate* operator) and sensing capabilities - meaning that fixed (F) and movable obstacles (M) will be the same for all manipulator robots. Since we want to experiment with our social cost model again, we will be experimenting using (Stilman, 2005)'s algorithm for the same reasons described in Section IV. 6.1. As such, all robots will operate under the same assumption of omniscient perception that

(Stilman, 2005)'s algorithm does, for the sake of simplicity:

$$\forall R_i \in R, W_{R_i} = W_{ref} \text{ and } \mathcal{A}_{R_i} = \mathcal{A}$$

$$\forall R_i \in R, F_{ref,R_i} = F_{ref} = F \text{ and } M_{ref,R_i} = M_{ref} = M$$

Implicit Communicating Multi-Robot System: We will also assume that only *implicit* (syn. Indirect) communication is allowed between the robots - that is, they can only obtain information about each other through environment observation and cannot directly send signals to one another. As discussed in Section II. 2.2.2, we consider that the first approach to Multi-Robot NAMO should ensure maximum robustness regarding the possibility of individual robot or communication failure. Consequently, the NAMO coordination strategy we propose in this chapter will thus need to be *fully implicit*.

One-to-one robot-obstacle relationship: We assume that movable obstacles may not be concurrently manipulated. At any given time t, an obstacle may only be manipulated by a single robot, in the same way that a robot may only manipulate one obstacle at once, which simplifies Equation V.2 by removing its last line:

$$\begin{aligned} &\exists R_i, R_j \in R, \exists t \in [\min(P_{R_i}, P_{R_j}), \max(P_{R_i}, P_{R_j})] \text{ s.t.} \\ &A_{R_i}(t, t+1) = Manipulate_{R_i}(W^t, M_k, G_m, \tau(q^t_{R_i}, q^{t+1}_{R_i})) \\ &\text{and } A_{R_j}(t, t+1) = Manipulate_{R_j}(W^t, M_k, G_n, \tau(q^t_{R_j}, q^{t+1}_{R_j})) \end{aligned} \tag{V.4}$$

This hypothesis strengthens the "One obstacle at a time" hypothesis of the NAMO literature, discussed in Section III. 1.4. As explained in Section II. 2.2.2, this hypothesis allows us to temporarily dodge the non-negligible difficulty of the Multi-Agent Object-Pushing Problem, notably making physics simulation simpler since we don't have to compute interactions on a same obstacle.

Synchronized discrete-time clocks: We assume time is discretized into simulation steps (as we have done until now), that are synchronized across robots (akin to a shared clock). Planning time is assumed to always fit within a simulation step, so that regardless of actual computation time, determinism of solutions is not affected.

It is to be underlined that the hypotheses of "Synchronized discrete-time clocks" and "Robot Omniscience" are compatible with that of a fully "Implicit Communicating Multi-Robot System". Actually, the omniscient hypothesis could be relaxed if we were to use (Wu&Levihn, 2014)'s algorithm instead of (Stilman, 2005)'s algorithm, so long as the robot sensing capabilities allowed it a field of view wide enough capable of guaranteeing absence of collisions. (Wu&Levihn, 2014)'s algorithm would also require a slight modification to its execution routine to trigger replanning not only when an obstacle intersects with the current plan, but also anytime an opening is observed, as to keep the local optimality guarantee. In any case, now

that our problem has been properly defined, let us now present our resolution proposition in the next section.

V. 2 Implicit Coordination in MR-NAMO

V. 2.1 Potential conflict detection

According to the problem definition in the previous section, a solution cannot allow conflicts. Thus, our algorithm must provide a means of detecting them; only then can robot adjust its plan as to avoid them. As stated in the same definition, conflicts can be straightforwarldy detected by comparing the plans of each robot that is part of the Multi-Robot System, and listing any action verifying any of the Equations V.1, V.3 or V.4. However, comparing plans would require explicit communication for the robots to share their plans [156] - which we do not allow.

As such, our robots may only *predict* **Potential Conflicts**, by comparing their remaining actions with the observed world state $W^t_{R_i}$. Our problem definition requires that potential conflict detection is to *guarantee* that it cannot fail to identify a situation as a potential conflict that will turn into an actual conflict (i.e. detection false negative). Identifying a situation as a potential conflict although it will not turn into an actual conflict (i.e. detection false positive) is however compatible with the problem definition. Still, potential conflict detection *should minimize false positives*, as they may imply subsequent planning, and thus computational overhead. That is why, in the following lines, we further differentiate our two main conflicts types, into 6 subtypes (abstractly synthesized in Fig.V.2 and discussed further in the following paragraphs) as to devise relevant detection strategies for each, that both guarantee absence of false negatives and reasonably limit false positives.

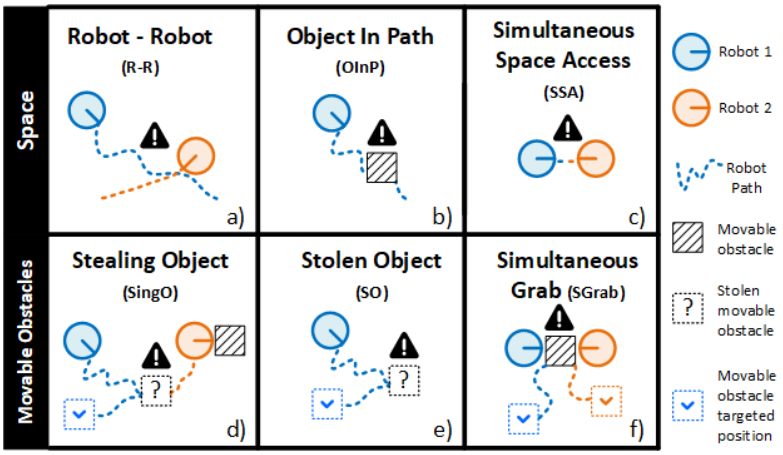


FIGURE V.2: Abstract representation of potential conflict types

Detecting potential conflicts consists in iterating over the remaining actions of the robot's plan after the current time t, and estimating whether executing each action at its planned time is likely to generate an actual conflict. Our hypothesis of "Synchronized discrete-time clocks" implies that it is first and foremost imperative to ensure that no conflict may happen between

the current time t and the next time step $t+1$. This is why we first define the following two potential conflict subtypes:

Simultaneous Space Access (abb. SSA) potential conflicts: the situations where two or more robots (or their eventual currently manipulated obstacles) could collide (i.e. try to access the same space) in between the current and next time step $[t, t+1]$, potentially verifying equation V.1. Without access to the other robots' plans, each robot R_i would need to compute the potential swept area for all possible actions for each other robot in their action space $\mathcal{A}_{R_j}$ within $[t, t+1]$, written as $S(\mathcal{A}_{R_j}(t, t+1))$. Computing the exact potential swept area is feasible under our "Homogeneous & Omniscient Multi-Robot System" hypothesis, since all robots share the same action space $\mathcal{A}$. However, we don't want our approach to be strongly tied to this hypothesis, nor spend the computation time it would require to simulate all possible actions for all other robots, at each time step. That is why we opt for a circular overestimate of $S(\mathcal{A}_{R_j}(t, t+1))$, as illustrated in Fig.V.3, accounting for any rotation or translation of each robot R_j that may occur within $[t, t+1]$. To guarantee this, the circle's radius must be at least equal to the radius of the circle circumscribing the robot R_j's polygon, plus the maximum translation expected in $\mathcal{A}$. *If the robot R_J is currently manipulating an obstacle M_k, the radius of the circle circumscribing the robot R_j's polygon* ***and*** *M_k's polygon is used instead* (Cf. Fig.V.3c).

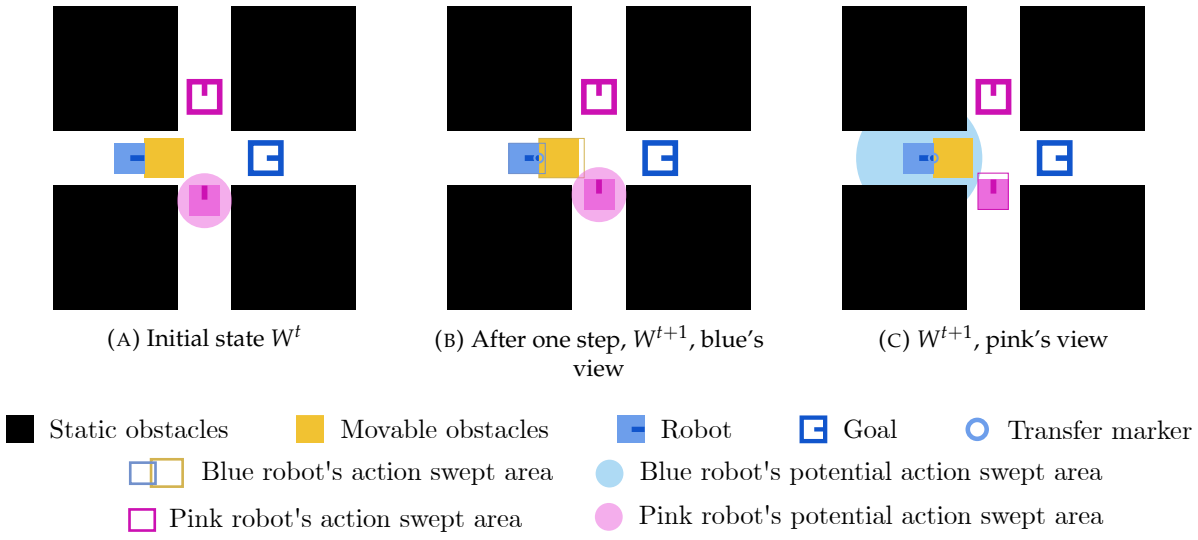
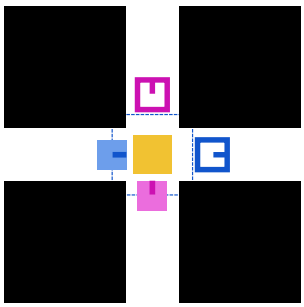


FIGURE V.3: Detection of a Simultaneous Space Access potential conflict. In (A), both robots compute plans. In (B) and (C), they have executed their first step: blue grabbed the obstacle (illustrated by transfer marker), while pink translated up. (B) and (C) respectively show the SSA potential conflicts detected by blue and pink, caused by the intersection of their next action's swept area with their circular overestimates of each other's next action.

Simultaneous Grab (abb. SGrab) potential conflicts: the situations where two or more robots could grab a same obstacle in between the current and next time step $[t, t+1]$, potentially verifying equation V.4. Again, each robot R_i can only predict whether another robot R_j is *likely* to grab the same obstacle M_k within $[t, t+1]$. We thus detect such potential conflicts under the following two conditions:

- R_j is not already transferring another movable obstacle M_l (it is otherwise formally impossible for R_j to grab M_k within $[t, t+1]$, because of the "One-to-one robot-obstacle relationship" hypothesis);
- R_j is within grabbing distance of M_k, which according to our NAMO problem definition in Section III. 1.4, is equal to the radius of the circle circumscribing the robot R_j's polygon, as illustrated in Fig.V.4. Again, according to our "Homogeneous & Omniscient Multi-Robot System" hypothesis, each robot R_i could compute the exact potential grasping configurations for each other robot R_j around obstacle M_k, but for the same reasons as for Simultaneous Space Access (abb. SSA) potential conflicts detection, we don't specialize our approach that much.



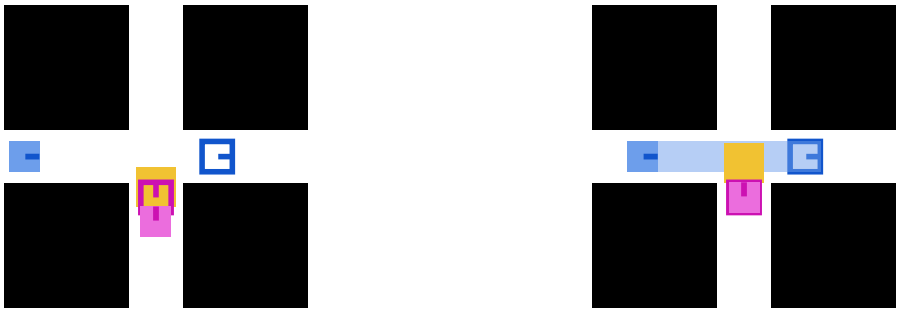
Grabbing distance-inflated polygon of obstacle to be moved by blue robot

FIGURE V.4: Detection of a Simultaneous Grab potential conflict. The center of each robot's is within the grabbing distance-inflated polygon of the movable obstacle, having them both consider that the other can instantaneously grab it within the next time step.

Detecting the above two subtypes of potential conflicts still is not sufficient to guarantee the absence of actual conflicts/collisions. For this, the three following situations also need to be detected:

Object In Path potential conflicts (abb. OInP): situations where a movable obstacle has been transferred (but is no longer being manipulated) on a robot R_i's path (Cf. Fig.V.5), potentially verifying equation V.1. Only movable obstacles that are not being transferred by a robot at time t need to be checked for intersection with the remaining plan steps.

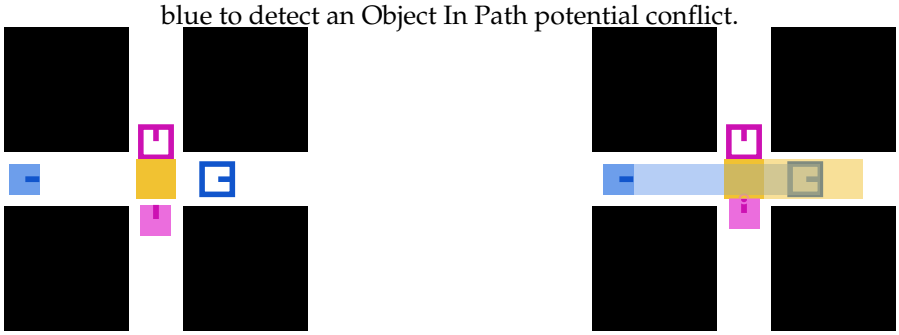
Stealing Object potential conflicts (abb. SingO): situations where a movable obstacle M_k that was planned for movement by a robot R_i is being transferred (Stealing) by another robot R_j before R_i (Cf. Fig.V.6), potentially verifying equation V.2. Under our hypothesis of an "Omniscient Multi-Robot System", we can directly check the grab status of the obstacle and detect the Stealing Object potential conflict type even before the obstacle is moved by the other robot R_j. Without this hypothesis, detecting Stealing Object potential conflicts could be done by observing the geometrical closeness between the other robot R_j and obstacle M_k and/or their speed vectors.



(A) Initial state W^t (B) After 5 steps, W^{t+5}

Blue robot's plan swept area

FIGURE V.5: Detection of an Object In Path potential conflict. (A) Both robots compute plans (pink's goal being partially covered by the robot and the movable obstacle). In (B), after 5 steps, pink reached its goal and released the obstacle, while blue translated right, causing blue to detect an Object In Path potential conflict.



(A) Initial state W^t (B) After 2 steps, W^{t+2}

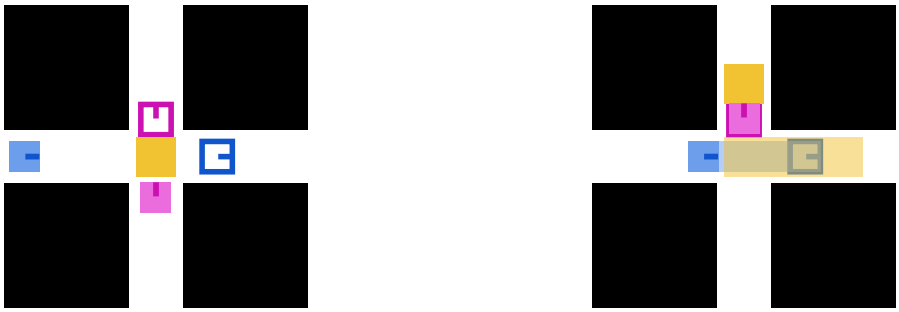
Blue robot's plan swept area

FIGURE V.6: Detection of a Stealing Object potential conflict. (A) Both robots compute plans. In (B), after 2 steps, pink grabbed the movable obstacle, while blue translated right wanting to grab the same obstacle, causing blue to detect a Stealing Object potential conflict.

Stolen Object potential conflicts (abb. SO): situations where a movable obstacle M_k that was planned for movement by a robot R_i has been transferred (Stolen) - *but is not being transferred* - by another robot R_j before R_i (Cf. Fig.V.7), potentially verifying equation V.2. Detection could be achieved simply by checking the grab status and whether the current movable obstacle configuration $q^t_{M_k}$ differs from the expected start configuration for the planned transfer path of robot R_i.

Since a movable obstacle cannot move by itself, these three subtypes of potential conflicts are very unlikely to solve themselves without the robot R_i taking appropriate action. Thus, the robot should detect these potential conflicts beyond the very short *horizon* of the next action to be executed within $[t, t+1]$, up until the current plan's end time t_{end}, similarly to (Wu&Levihn, 2014)'s execution routine (Alg.1). This discussion about the detection horizon brings us to our last subtype of potential conflict:

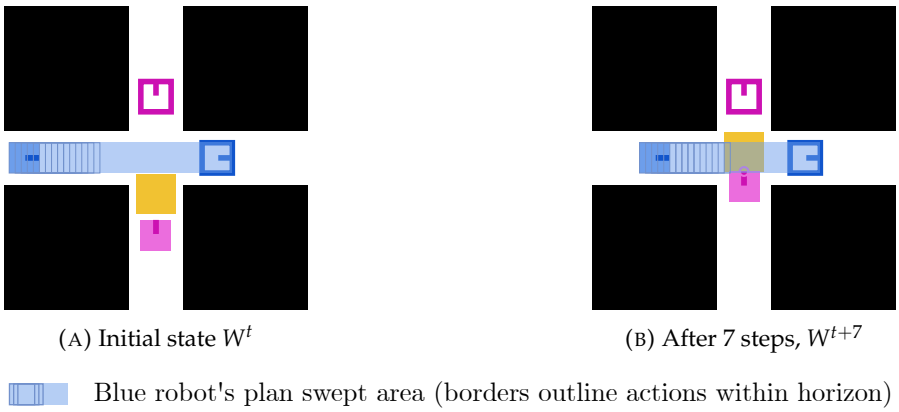
Robot-Robot potential conflicts (abb. R-R): situations where, in between $[t, t+h]$, $h > 1$ being a given time horizon, the plan of one robot R_i geometrically intersects with another robot R_j or their eventual currently manipulated obstacle (Cf. Fig.V.8), potentially verifying equation



(A) Initial state W^t (B) After 15 steps, W^{t+15}

FIGURE V.7: Detection of a Stolen Object potential conflict. (A) Same initial situation as Fig.V.6a, both robots compute plans. In (B), after 15 steps, pink transferred and released the movable obstacle, while blue translated right wanting to grab the same obstacle, causing blue to detect a Stolen Object potential conflict. This scenario happens under the assumption that the robot does not react subsequently to the detection of previously detected Stealing Object potential conflicts.

V.1. They can be understood as an extension of Simultaneous Space Access potential conflicts beyond the next time step $t+1$, allowing the robot to react sooner than at the last step. It would be however be counter-productive to have the robot react every time another robot temporarily crosses its plan so far away that by the time the robot gets there, the other one is already long gone - hence the introduction of the check horizon h. The smaller h, the less often the robot needs to react to other robot's movements, but the later it can react thus uselessly augmenting the robot displacement cost to the goal.



(A) Initial state W^t (B) After 7 steps, W^{t+7}

FIGURE V.8: Detection of a Robot-Robot potential conflict. (A) Both robots compute plans, but for readability, only blue's plan is shown with the swept area of its next 10 actions. In (B), after 7 steps, pink is transferring the movable obstacle, while blue translated right, causing blue to detect a Robot-Robot potential conflict as pink and its movable obstacle's footprints intersect with blue's horizon.

There as many potential conflicts to a robot's plan $\mathcal{P}$ in a world state W^t as there are intersections with the collision geometries we just presented. Even a single action's swept area may intersect with multiple other entities, generating a potential conflict detection for each. The whole point of detecting these potential conflicts is to allow the robot to take a decision as to what to do in these crucial moments, in order to prevent these potential conflicts

from becoming actual conflicts. We shall discuss this decision process in the following sections.

V. 2.2 Conflict avoidance: a timing-based strategy

In order to prevent these previously defined potential conflicts from actually happening, we drew inspiration from the Fixed-path Coordination described by S. Lavalle in his reference book on Motion Planning [154]. It consists in planning for each robot using a single-robot algorithm, then schedule the motion of the robots along the plans so that they do not collide by tuning execution speed, or introducing waiting times if speed is constant, as is our case. Thus, once a robot has computed its NAMO plan, we broadly coordinate the execution as follows, at each time step (formalized as pseudocode in Algorithm 10, and alternatively, as a decision graph in Fig.V.9):

1. Potential conflicts are detected at a **variable horizon** depending on potential conflict type (Alg. 10, l.7): the DETECT-POTENTIAL-CONFLICTS function returns a list of potential conflicts, since there are as many potential conflicts as there are intersections with collision geometries, as explained in the previous section;

2. If a potential conflict is detected, the current plan is **postponed for a random time** within a pre-decided interval (Alg. 10, l.9);

   - if potential conflicts persist until the postponement's end, **re-planning** is triggered (Alg. 10, l.10),
   - otherwise, plan execution is resumed if they disappear during the postponement (Alg. 10, l.8).

3. If a Stolen Object (SO) or Object In Path (OInP) potential conflict is among the detected potential conflicts, re-planning is immediately triggered (Alg. 10, l.10), as they are very unlikely to disappear simply by having the robot wait, since movable obstacles can't move by themselves.

This straightforward and fully-implicit strategy constitutes a first approach to solving MR-NAMO problems without explicit communication. We discuss and argument the components of this strategy in the following subsections.

Algorithm 10: Coordination NAMO (C-NAMO) Algorithm

Data: World state W^t, Goal q_R^g, Current plan $\mathcal{P}$, Minimum and maximum number of steps for timer $[t_{min}, t_{max}]$, Maximum number of NAMO-PLAN calls n_{try}, Robot-Robot Conflict detection horizon h

Result: Action A^t to execute between t and $t+1$, Current plan $\mathcal{P}$

while A^t is not SUCCESS **or** FAILURE **do**
  $W^t \leftarrow$ GET-NEW-INFORMATION();
  $A^t, \mathcal{P} \leftarrow$ C-NAMO(W^t, q_R^g, $\mathcal{P}$, t_{min}, t_{max}, n_{try}, h);
  TRY-TO-EXECUTE(A^t)

C-NAMO(W^t, q_R^g, $\mathcal{P}$, t_{min}, t_{max}, n_{try}, h)
  if $q_R^t = q_R^g$ **then return** SUCCESS, $\mathcal{P}$;
  $C \leftarrow$ DETECT-POTENTIAL-CONFLICTS($\mathcal{P}$, W^t, h)
  if $\mathcal{P} \neq \emptyset$ **and** $C = \emptyset$ **then return** NEXT-STEP($\mathcal{P}$), $\mathcal{P}$;
  else if $\mathcal{P} \neq \emptyset$ **and** $C \neq \emptyset$ **and** SO or OInP conflict $\notin C$ **then return** POSTPONE($\mathcal{P}$, t_{min}, t_{max}) ;
  else return REPLAN($\mathcal{P}$, W^t, q_R^g, t_{min}, t_{max}, n_{try}, h) ;

REPLAN($\mathcal{P}$, W^t, q_R^g, t_{min}, t_{max}, n_{try}, h)
  if TRIES-LEFT($\mathcal{P}$, n_{try}) **then**
    $W^t_{noD} = \{\forall E_i^t \in W^t | E_i^t \notin R, M*\}$;
    $\mathcal{P} \leftarrow$ NAMO-PLAN(W^t_{noD}, q_R^g);
    $C \leftarrow$ DETECT-CONFLICTS($\mathcal{P}$, W^t, h);
    if $\mathcal{P} \neq \emptyset$ **and** $C = \emptyset$ **then return** NEXT-STEP($\mathcal{P}$), $\mathcal{P}$;
    else if $\mathcal{P} \neq \emptyset$ **and** $C \neq \emptyset$ **and** TRIES-LEFT($\mathcal{P}$, n_{try}) **then**
      $W^t_D = W^t_{noD} \cup \{$conflicting entities in $C\}$;
      $\mathcal{P}' \leftarrow$ NAMO-PLAN(W^t_D, q_R^g);
      $C \leftarrow$ DETECT-CONFLICTS($\mathcal{P}'$, W^t, h);
      if $\mathcal{P}' \neq \emptyset$ **and** $C = \emptyset$ **then**
        $\mathcal{P} \leftarrow \mathcal{P}'$;
        return NEXT-STEP($\mathcal{P}$), $\mathcal{P}$;
      else return POSTPONE($\mathcal{P}$, t_{min}, t_{max});
  return FAILURE, $\mathcal{P}$

POSTPONE($\mathcal{P}$, t_{min}, t_{max})
  if IS-TIMER-OFF($\mathcal{P}$) **then** START-RANDOM-DURATION-TIMER($\mathcal{P}$, t_{min}, t_{max}) ;
  else if IS-TIMER-OVER($\mathcal{P}$) **then return** REPLAN($\mathcal{P}$, W^t, q_R^g, t_{min}, t_{max}, n_{try}, h) ;
  return WAIT-ACTION, $\mathcal{P}$;

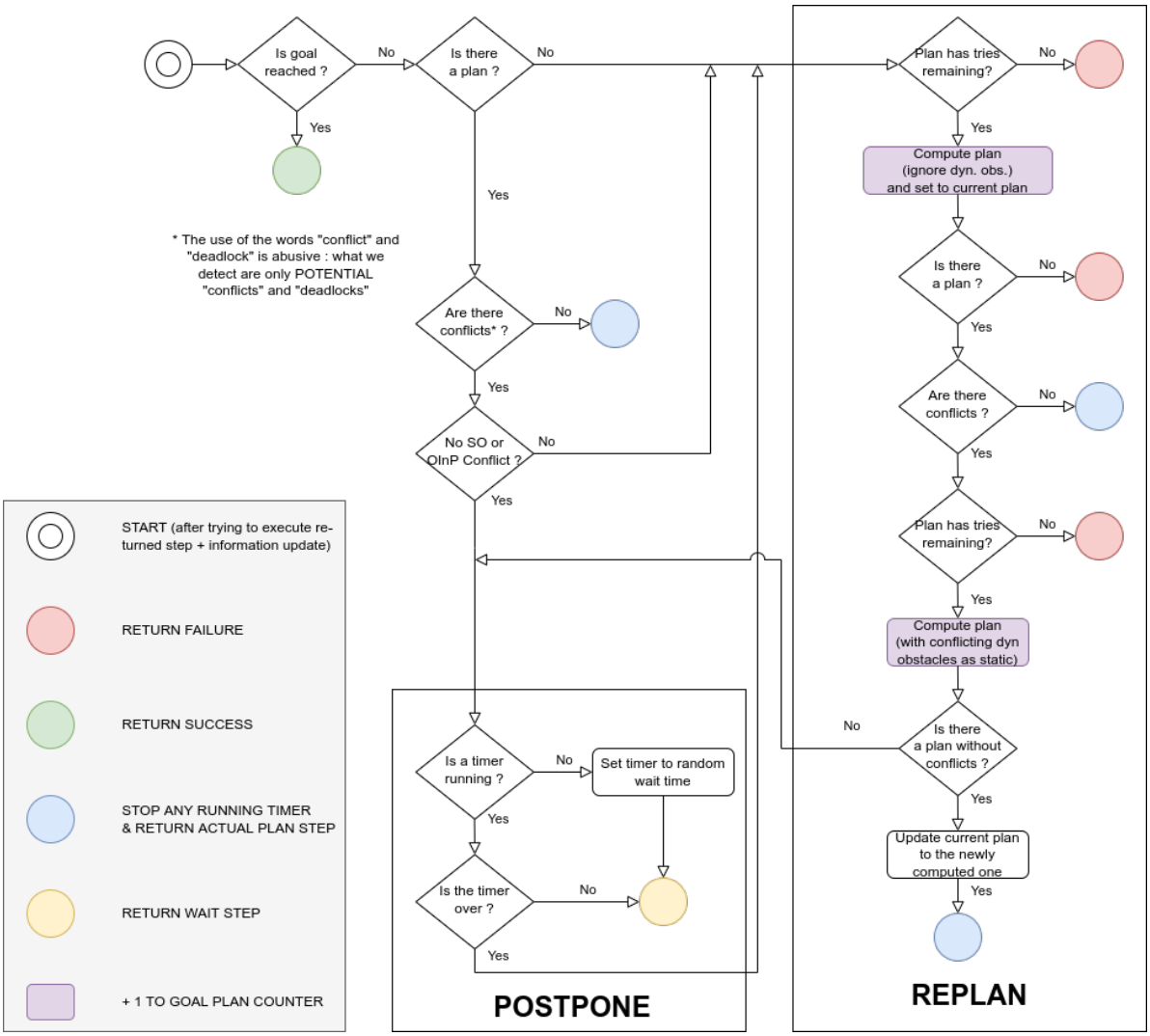


FIGURE V.9: Implicit coordination strategy decision graph for potential conflict resolution, equivalent to Algorithm 10, C-NAMO.

V. 2.2.1 Postponement

Upon potential conflict detection, the robot only has two options: updating the plan (Alg. 10, l.10) or waiting in hope that the potential conflicts disappear on their own (Alg. 10, l.9). The latter consists in starting a timer for a random duration uniformly selected within a reasonable interval $[t_{min}, t_{max}]$ (Alg. 10, l.27). This timer is preempted (and thus, reset) if the potential conflicts disappear (Alg. 10, l.8), otherwise, a re-planning is triggered once the timer is over (Alg. 10, l.28 - the timer is also reset upon the IS-TIMER-OVER function returning True). It is to be noted that, as the formal writing suggests in Alg. 10, the timer is plan-dependent: once the plan is updated, the timer is reset too.

The randomness allows to break the symmetry in potential conflicts like Simultaneous Space Access (Cf. Fig.V.3) and Simultaneous Grab (Cf. Fig.V.4), or mutual Robot-Robot potential conflicts where two robots are progressing towards one another and would cause potential conflicts detections at every step. With a random duration, one robot will almost always wait a shorter amount of time, re-plan before the other and seize the space or obstacle first.

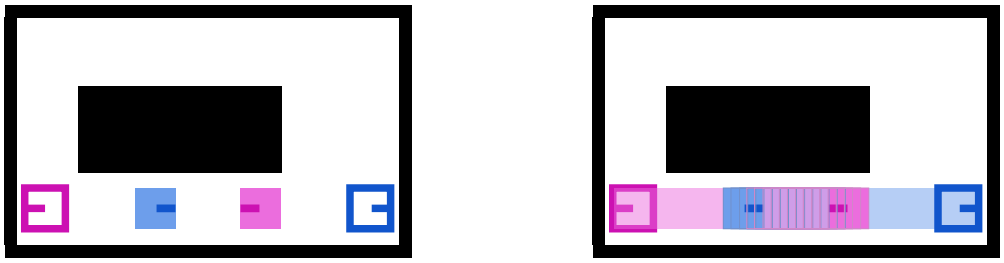
Lastly, Object in Path (OInP) and Stolen Object (SO) potential conflicts are very unlikely to disappear simply by having the robot wait, since movable obstacles can't move by themselves. That is why we immediately re-plan whenever they are detected (Alg. 10, l.9). One could expect we would do the same for Stealing Object conflicts, but it actually makes more sense to not immediately re-plan, but rather wait so that re-planning occurs once the obstacle has been cleared by another robot.

Finally, in order to guarantee that the algorithm terminates, the number of calls to the NAMO-PLAN function is limited to a ceiling value n_{try} by the TRIES-LEFT function (Alg. 10, l.12, 17). This value is chosen as to balance experiment time and experiment resolution capability: ideally the robot needs sufficient tries to solve the given problem within the time available to the experimenter.

V. 2.2.2 NAMO planning in a dynamic environment

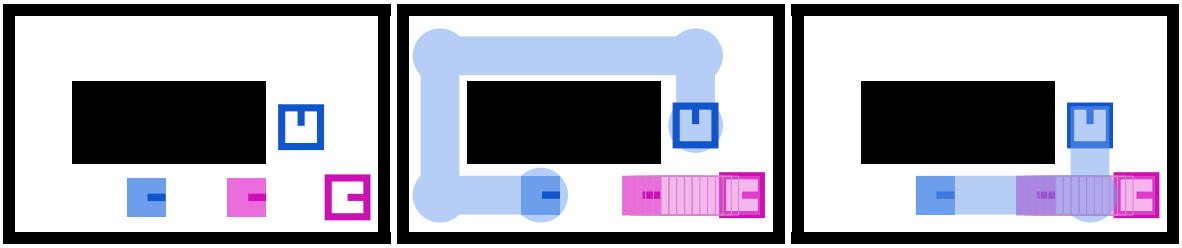
Existing NAMO planners assume that there can only be two types of entities in the environment beyond the single robot manipulator: movable and static obstacles. Hence, when using such planners as-is in a dynamic environment with other agents such as robots (i.e. dynamic obstacles), one needs to decide how to identify these new entities as per the NAMO classification. Since we made the hypothesis that a robot cannot be manipulated by another as a movable obstacle, we are left with only two choices: either ignore dynamic obstacles when using an existing NAMO planner, or classify them as static obstacles.

On the one hand, always ignoring dynamic obstacles all the time would lead to the robot desperately trying to compute the same plan going through another robot's footprint over and over again, ignoring other obvious but slightly more convoluted possibilities that may lead to the goal (Cf. Fig.V.10). On the other hand, always considering dynamic obstacles as static obstacles during the NAMO planning procedure may lead to unnecessary detours (Cf. Fig.V.11), or unsolvable situations (Cf. Fig.V.12).



(A) Initial state W^t (B) Robot plans ignoring each other's footprint

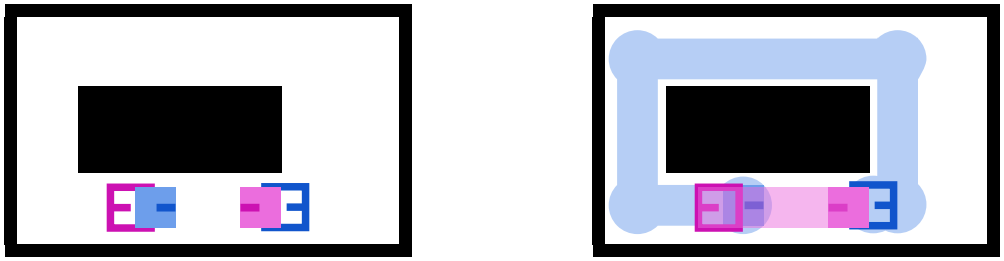
FIGURE V.10: Example deadlock situation between two robots facing one another, caused by their systematically ignoring each other during planning, even though an alternative path is available.



(A) Initial state W^t (B) Inefficient plan when dynamic obstacles are always considered static (C) Ideal plan if robot just waited a bit longer

FIGURE V.11: Example situation where the blue robot takes a huge detour, caused by the systematic consideration of the other robot as a static obstacle.

In order to try to avoid most of these situations without having to fundamentally change the NAMO planner used in the coordination strategy, we have the robot do both. Upon **REPLAN** (Alg. 10, l.11-25), a NAMO plan is first computed by ignoring all dynamic obstacles (Alg. 10, l.13-14). If no plan is found, we assume then that the goal must fail because the NAMO problem is just too complex for the NAMO planner. If we detect potential conflicts with this first plan (Alg. 10, l.15-16), we try to recompute a plan by considering the conflicting dynamic obstacles as static obstacles (Alg. 10, l.18-19). If no valid plan is then found, the plan that ignored dynamic obstacles is postponed (Alg. 10, l.24), in the hope that the conflicts resolve themselves while the robot waits.



(A) Initial state W^t (B) The ideal solution

FIGURE V.12: Example deadlock situation where the two robots face each other while occupying their respective goals, never succeeding in computing a plan as they both systematically consider each other to be static obstacles.

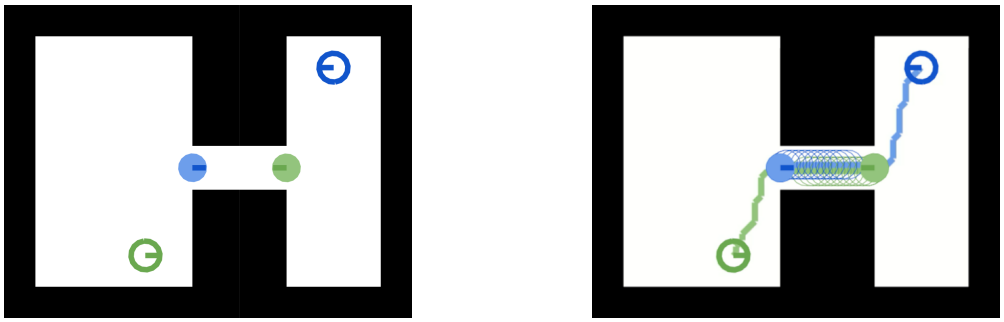
V. 2.3 Deadlock evasion

V. 2.3.1 Detecting potential deadlocks

While the previously formulated coordination strategy guarantees that the problem's constraints cannot ever be violated, it may result in **deadlocks - that is, dependency cycles between robots' actions** [156]. Deadlocks are intrinsic problems of decentralized algorithms, as they emerge from the dependency cycles that appear in the merging process of independently computed decision trees. For Multi-Robot Systems, they can either cause the Robot Freezing Problem (i.e. the robots involved in the deadlock can not move) or oscillations (i.e. the robots involved in the deadlock indefinitely alternate between the same configurations). In an explicitly communicating multi-robot system, deadlocks can be straightforwardly solved by sharing robot plans, detecting the cycles in the shared dependency graph of robot actions and centrally planning a resolution for all robots.

Without explicit communication, in the same way we explained in Section V. 2.1 that robots could only detect potential conflicts, they can also only detect **Potential Deadlocks**. However, while our problem definition requires that no conflict actually occur, it does not require the non-occurrence of deadlocks. As such our potential deadlock detection strategy is allowed both false negatives or positives, but should still minimize both.

While the actual dependency cycles between plans cannot be observed, it is still possible for each robot to observe and remember the occurrence of potential conflict detections, as defined in Section V. 2.1. **We thus define a potential deadlock as the redundancy of a potential conflict over time**: that is, if a robot in the execution process of reaching a same goal encounters a same-type potential conflict, occurring at the same configuration of all robots involved, a potential deadlock is detected, as shown in Fig.V.13.



(A) Initial state W^t (B) Robot plans

FIGURE V.13: Two robots in a deadlock, because both detect a Robot-Robot potential conflict involving the other.

If the time period to consider a recurring potential conflict is too small, then false positives may be too frequent and result in the robots trying to avoid deadlocks that do not exist. Since we assume that all robots run with the same algorithm and parameters, **we wait for the maximum duration of a postponement to pass before considering a recurring conflict to be a deadlock**. Under a different hypothesis, one may have to more carefully consider what a good time span would be depending on the hypothesis.

V. 2.3.2 Evading potential deadlocks

Once a potential deadlock has been detected by a robot, it must decide whether to move or not, and if so, where to, in order to try and resolve the situation. As explicit communications are forbidden in our problem, the robot could for example randomly decide to move or not, just as it could try to reach a random pose in the environment or try to drive backward in a straight line, and rinse and repeat until maybe the deadlock gets resolved. That would of course be woefully naive and inefficient, both in terms of displacement and computational costs, if at all useful.

But when you think about it, **in a deadlock situation, the robot itself is a (dynamic) obstacle in the other involved robot(s)' view; and deciding whether to give way and how best to do so, does sound a lot like a social navigation problem**. Then, in the same way our Social Occupation Cost model highlights relevant positions for movable obstacles that improve space accessibility, it could help robots decide if and where to go to in these situations. The broad steps of this procedure are as follows:

1. Potential deadlocks are detected as exposed in the previous section;
2. If a potential deadlock is detected, the robot computes an "evasion plan" for itself; the destination of which being the configuration with least combined social and distance cost;
3. To decide whether to use this evasion plan:
   (a) The robot also computes the "potential evasion plans" of the other robots involved in the deadlock;
   (b) If the robot's evasion configuration social occupation cost is the highest among all the computed evasion plans (i.e. most unlikely to solve the deadlock), the robot postpones its current plan (waiting for the others to move), otherwise, it executes its own evasion plan. This way, only one robot should wait while the others evade.

This process, which we will further discuss and illustrate in the following lines, is integrated into the previously discussed coordination strategy, as shown in Fig.V.18 and Algorithm 11, obtained within our simulator.

Evasion plans are each computed using Dijkstra's algorithm on the grid inflated by the robot's circumscribed circle radius. The evasion configuration is chosen similarly to the obstacle placement in the previous chapter, by combining the displacement cost obtained using the Dijkstra search and the social occupation cost, but not the displacement cost estimate to the goal (since the robots do not share their respective goals). The path is then straightforwardly constructed from this evasion configuration by traversing the Dijkstra search tree. This process is illustrated in Figures V.14 A to C.

We choose to assume that robots will drop their transferred obstacle before executing the evasion plan, if they choose to evade, for the sake of simplicity. Otherwise, each robot would not only need to compute their plans while accounting for collisions for transferred obstacles,

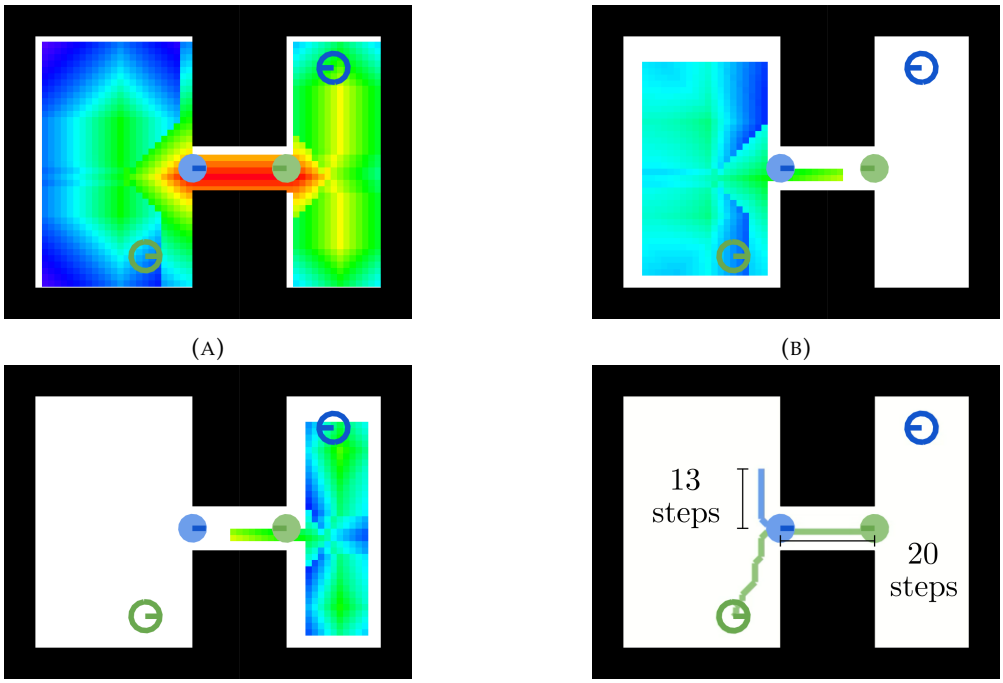
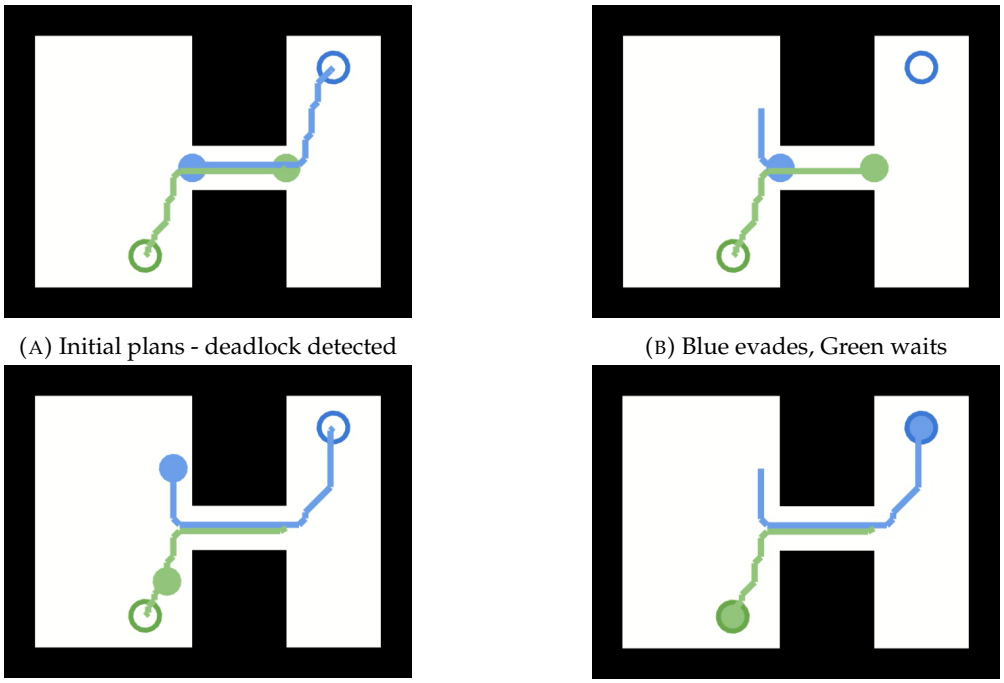


FIGURE V.14: Illustration of the blue robot's deadlock evasion planning process after detection in Fig.V.13. (A) The corresponding social occupation costmap. (B) The combined costmap of the blue robot's evasion plan, which corresponds to Dijkstra search tree - the greener, the higher the cost. (C) Same illustration for the green robot's potential evasion plan computed by the blue robot. (D) The resulting evasion plan for blue, while green chooses to wait for blue to move; the number of steps of the blue robot's evasion plan (13) and of the green robot's would-be plan to reach the blue robot's current configuration (20) are added as wait time after evasion for the blue robot to let green pass before resuming.

but it would also need to consider the social occupation cost of the obstacles, resulting in computational overhead.

Upon reaching the evasion configuration, the robot needs to wait for a moment so that the other robots involved in the potential deadlock may have time to move themselves, hopefully in a way that will solve the deadlock. Carefully choosing this waiting time is essential in order to augment the chances for the robot to solve the deadlock. For this purpose, we consider the time it would take for the other involved robot that is furthest from the evading robot to reach the same evasion configuration. This time is the sum of the evading robot's evasion plan time, and the time of a navigation plan from its current position's to the furthest robot. The first one has already been computed, and the second one can be obtained by A-Star searches towards the other involved robots' current positions. The computation is illustrated in Figure V.14d, and the full evasion process including the wait is showcased in Figure V.15.

This extra waiting time upon evasion completion is applied by executing WAIT steps that are directly appended to the evasion plan: this makes sure that the wait is not preempted as soon as conflicts disappear, triggering re-planning too early and causing deadlocks all over again. Because deadlocks are worst-case scenarios for plan execution that can lead to goal failure, full evasion completion takes full priority.



(A) Initial plans - deadlock detected

(B) Blue evades, Green waits

(C) Blue re-plans once its mandatory wait has passed

(D) Final state - deadlock solved

FIGURE V.15: Complete deadlock resolution of Fig.V.13's scenario.

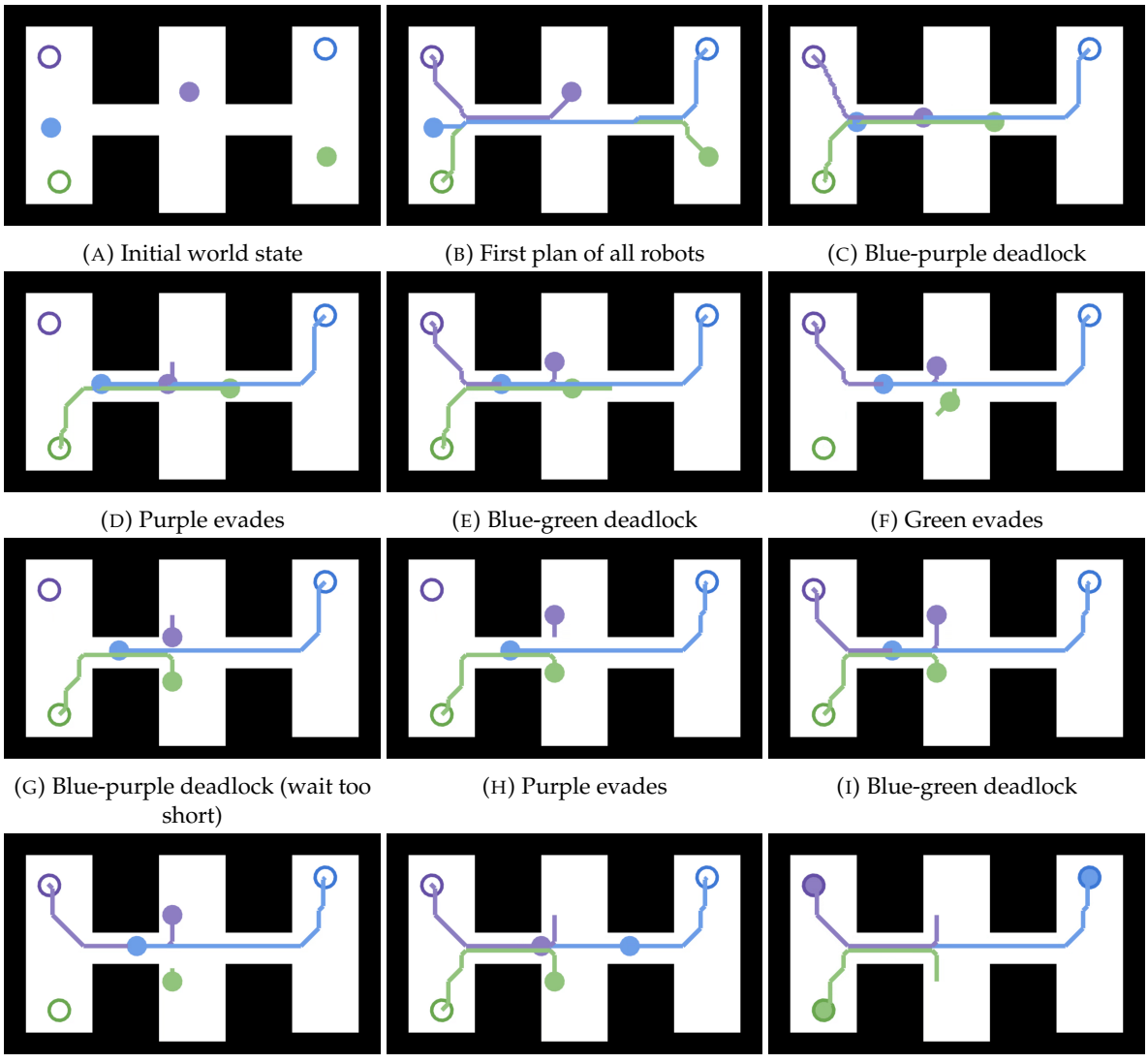
Again, we remind that our coordination only guarantees that potential conflicts never actually become actual one, but our deadlock evasion strategy does not guarantee resolution: such a guarantee would either require explicit communication or artificial restrictions to specific scenarios. **While our method works best under our hypothesis of Homogeneous & Omniscient Multi-Robot System, it is not a fundamental requirement**. Indeed, since our method relies on predicting the other robot's behavior, the more accurate the prediction, the higher chance of deadlock resolution.

In order to qualitatively showcase this deadlock evasion strategy, we added snapshots of the different stages of a three-robots scenario illustrated in Figure V.16, and a two-robots NAMO scenario with a single movable obstacle, illustrated in Figure V.17. We did not have the opportunity to use this deadlock evasion strategy in the large-scale quantitative experiments showcased in the following Section V. 3.

In the three-robots scenario (Cf. Fig.V.16), the robots need to go from one room to the other through narrow corridors, which results in the creation of multiple successive deadlocks. As the robots sometimes underestimate the wait time after evading to let other robots pass, they resume their plans slightly too soon (Cf. Fig.V.16g & V.16h), resulting in the reappearance of a similar deadlock, needing evasion again. However, this process eventually converges, as **we forbid the robot to use the same evasion configuration twice for the same goal, in order to prevent oscillation**; this can be observed with the purple robot between Fig.V.16d to Fig.V.16j.

In the two-robots NAMO scenario (Fig.V.17), the robots again need to go from one room to the other through a narrow corridor, but this one is blocked by a movable obstacle. As the blue robot grabs the obstacle first, the green robot abandons its initial NAMO plan for a pure

navigation plan, eventually blocking the rightmost room's entrance (Cf. Fig.V.17a to V.17e). This results in a deadlock, which the green robot evades several times as to finally free the blue robot's path (Cf. Fig.V.17f to V.17j). The blue robot finally resumes its NAMO plan, leaving the obstacle in an arguably not so good position(Cf. Fig.V.17k to V.17l), as the green robot blocked the way to the "recesses" of the rightmost room's walls at the time of the blue robot's planning. The green robot thus has to move the obstacle again to get to its goal (Cf. Fig.V.17m to V.17o). Had the robots been allowed to communicate their goals and plans, maybe a better obstacle choice could have been selected in regard to their respective objectives, such as the top-left room corner. While showcasing the resolution capabilities of our approach, this example also shows a limit of a fully-implicit coordination strategy in solving MR-NAMO problems.



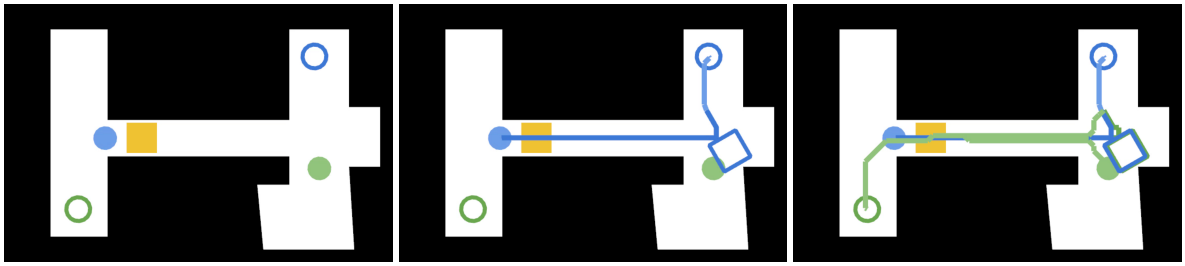
(A) Initial world state (B) First plan of all robots (C) Blue-purple deadlock

(D) Purple evades (E) Blue-green deadlock (F) Green evades

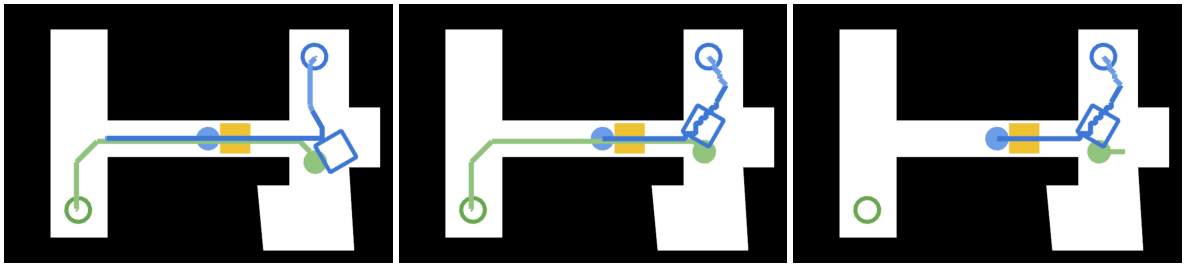
(G) Blue-purple deadlock (wait too short) (H) Purple evades (I) Blue-green deadlock

(J) Green evades (K) Purple resumes first (L) Green resumes, problem solved

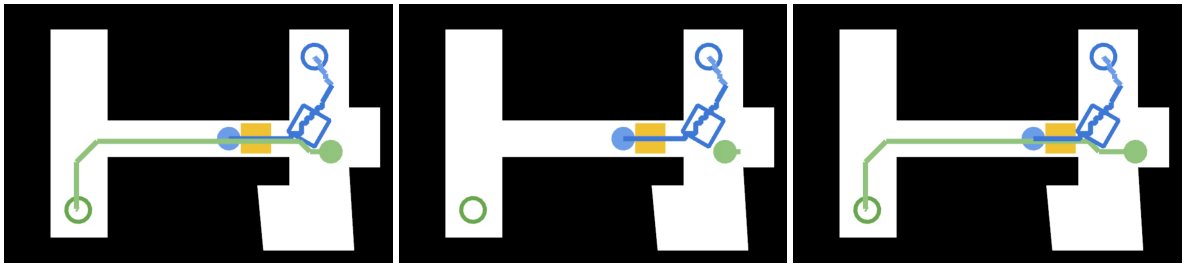
FIGURE V.16: Three-robot scenario showcasing successive deadlock situations. This example showcases how even if the first deadlock is not solved "in one go", our evasion strategy still makes relevant decisions that quickly converge to a resolution.



(A) Initial world state (B) Blue robot first plan (C) Green robot first plan

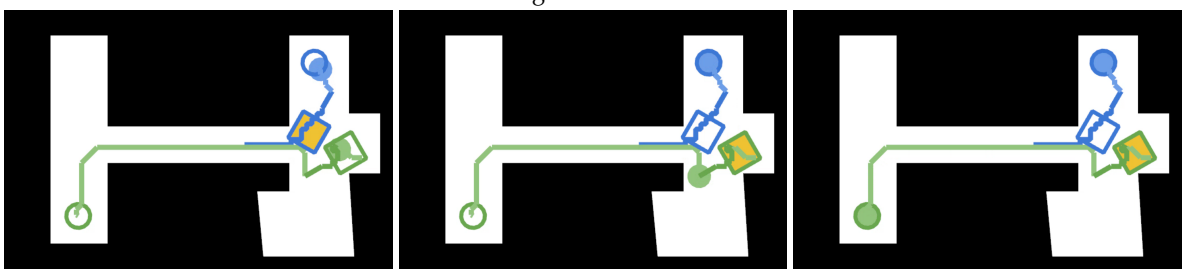


(D) R-R potential conflict detected by green, which replans (E) R-R potential conflict detected by blue, which replans, but deadlock (F) Green evades



(G) Green resumes plan after evasion, but deadlock is not solved (H) Green evades again (I) Again, green resumes plan after evasion, but deadlock is not solved

(J) Green finally evades far enough (K) Blue executes its plan while green waits (L) Blue releases the obstacle



(M) Green replans (N) Blue reaches its goal (O) Green reaches its goal

FIGURE V.17: MR-NAMO scenario showcasing successive deadlock situations. As in Fig.V.16, this example also showcases how the deadlock is progressively solved.

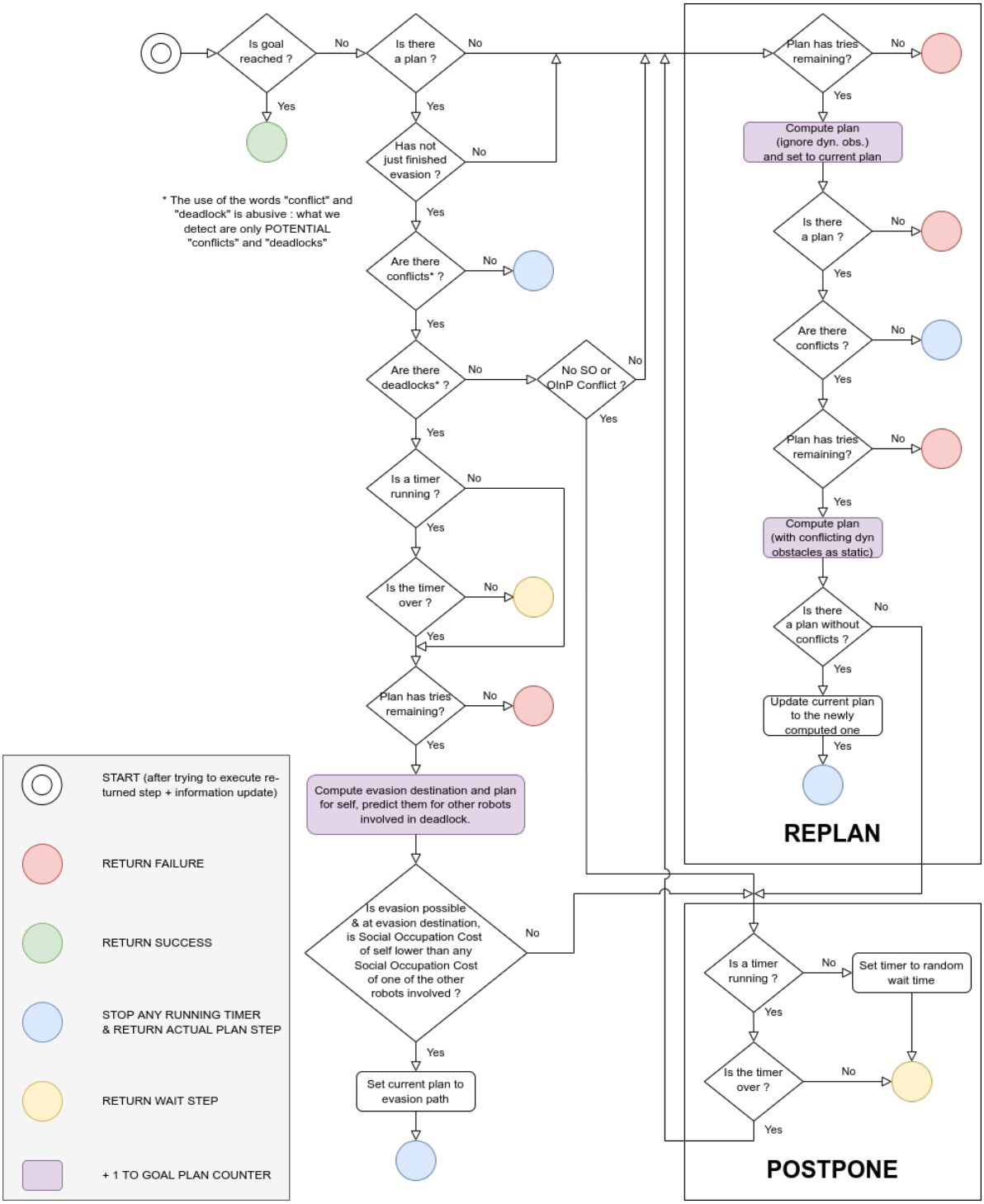


FIGURE V.18: Implicit coordination strategy decision graph augmented with potential deadlock resolution, equivalent to Algorithm 11, C-NAMO+

Algorithm 11: Improved Coordination NAMO (C-NAMO+) to evade deadlocks

Data: World state W^t, Goal q_R^g, Current plan $\mathcal{P}$, Minimum and maximum number of steps for timer $[t_{min}, t_{max}]$, Maximum number of NAMO-PLAN and EVASION-PLAN calls n_{try}, Conflict detection horizon h

Result: Action A^t to execute between t and $t+1$, Current plan $\mathcal{P}$

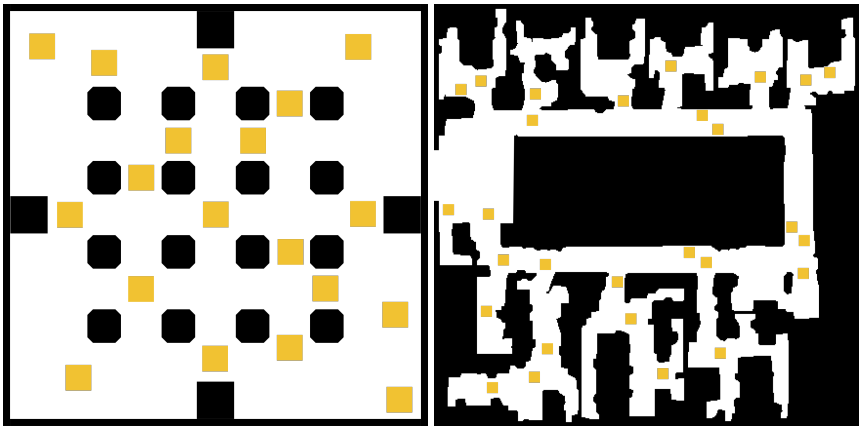
```
C-NAMO+(W^t, q_R^g, P, t_min, t_max, n_try, h)
    if q_R^t = q_R^g then return SUCCESS, P;
    if P = ∅ or IS-EVASION-OVER() then return REPLAN(P, W^t, q_R^g, t_min, t_max, n_try, h);
    C ← DETECT-POTENTIAL-CONFLICTS(P, W^t, h);
    D ← DETECT-DEADLOCKS(P, C);
    if C = ∅ then return NEXT-STEP(P), P;
    if D = ∅ then
        if SO or OInP conflict ∉ C then return POSTPONE(P, t_min, t_max) ;
        else return REPLAN(P, W^t, q_R^g, t_min, t_max, n_try, h) ;
    else
        if IS-TIMER-OFF(P) OR IS-TIMER-OVER(P) then
            if TRIES-LEFT(P, n_try) then
                P' ← EVASION-PLAN(W^t);
                if P' ≠ ∅ then
                    P ← P';
                    return NEXT-STEP(P), P;
                else return POSTPONE(P, t_min, t_max);
            else return FAILURE, P;
        else return WAIT-ACTION, P;
```

V. 3 Experiments

In the previous section, we explained the challenges of using existing NAMO algorithms in a Multi-Agent environment, discussed an implicit coordination strategy to address them, and qualitatively justified our choices with simple and specific problem examples. This section aims to:

- study the scalability of our approach as the number of robots grows;
- measure the impact of our Social Occupation Cost Model in a multi-robot NAMO context.

V. 3.1 Experimental context



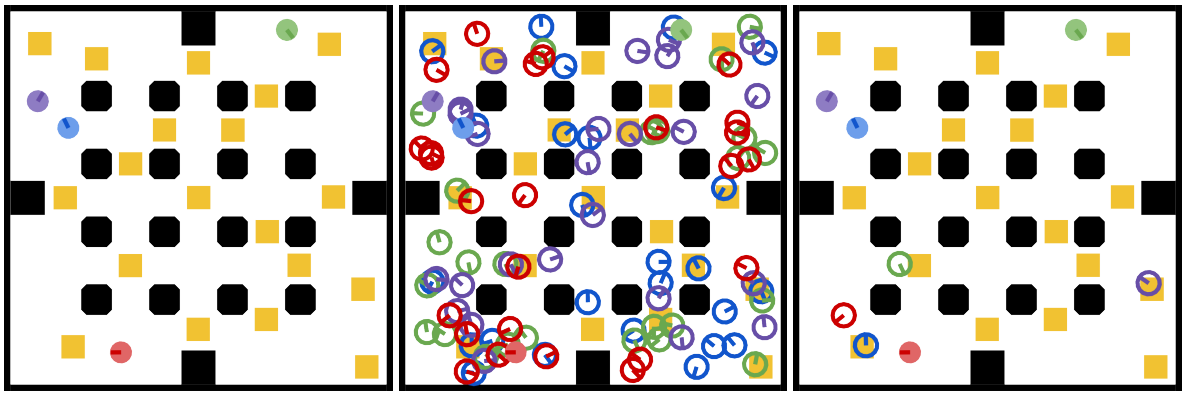
(A) Intersections
(abb. Int, env. 9.5x9.5 m)

(B) CITI Laboratory
(abb. CITI, env. 22.5x22.5 m)

FIGURE V.19: Base environments for scenarios. Static obstacles in black, movable in yellow.

We reused the two large-scale environments introduced in Chapter IV, re-illustrated in Fig.V.19: "Intersections" and "CITI Laboratory". On the one hand, the "Intersections" scenario is fully handcrafted to offer multiple intersections next to each other, since we previously identified that they are especially challenging for NAMO algorithms. On the other hand, the "CITI Laboratory"' scenario is a real world occupancy grid of our lab, with a greater area and where intersections are further away from one another.

We generated **200 scenarios for 2 robots in each of these environments** (abb. Int-2 and CITI-2), and **200 scenarios for 4, 5 and 10 robots** (abb. Int-4, Int-5, Int-10) **in the "Intersections" environment** (for a grand total of 1000 scenarios). Similarly to Section IV. 7.3.2, each scenario consists in randomizing the initial robots' poses and their respective goal sequence. The number of goals for a given scenario is fixed at 100, evenly distributed among the number of robots (e.g. 2 robots will each have 50 goals, 4 robots 25 goals, etc.), in order to keep measurements such as planning time comparable across the different number of robots. Figure V.20 illustrates one of the 4-robot scenarios, with its randomly generated initial and goal robot configurations.



(A) Initial robot configurations only (B) All 100 goals (C) First goal of each robot only

FIGURE V.20: Example of a 4-robot scenario, with 25 goals per robot.

The values of the parameters that are specific to the implicit coordination strategy are synthesized in the table below. All other parameters related to Stilman's algorithm, and to the social occupation cost model used for obstacle placement choice or robot evasion configuration choice stay the same as in the experimental section of Chapter IV.

Maximum number of NAMO/Evasion planners calls n_{try}	10
Random timer duration interval $[t_{min}, t_{max}]$	[5, 20]
R-R Conflict detection horizon h	10

TABLE V.1: Coordination algorithm parameters values for the experiments

The scenarios executions were distributed across 64 machines with Intel Core i7-4790 CPUs (3.60GHz - 4 cores/8 threads) and 8 GB of RAM as individual processes, one process per thread. The different runs occur within similar CPU usage at all times, keeping planning times comparisons relevant. Also, since simulation time steps are synchronized across robots, varying plan computation times depending on machine CPU usage will not affect the determinism of the plan's computations. The random seed used for the random durations timers generation is saved as a scenario parameter also to guarantee determinism and experiment reproducibility.

V. 3.2 Evaluation criteria

Now that we are considering several robots instead of one, robot-dependent evaluation criteria need to be aggregated together in a meaningful fashion, and separated from world-dependent (i.e. robot-independent) criteria that don't need to be aggregated. The social-aware criteria presented in Chapter IV are unaffected by the multi-robot nature of the problem, and are world-dependent criteria. When it comes to robot-dependent criteria, according to Stern et al.'s survey [106], the two most frequent objective functions that Multi-Agent Path Finding algorithms try to minimize are the Makespan (the cost of the costliest individual robot plan) and the Sum of Costs (of all the individual robots' plans); which is why we use the following evaluation criteria:

- **Execution time Makespan t_{max} [steps]**: the number of simulation steps for all robots to traverse their individual goal sequences; since time is the cost measure of choice in Multi-Agent problems, because of the possibility for the robots to have to vary speed or wait to prevent collisions and deadlocks (which does not affect the traversed distance).
- **Sum of Traversed Distances L_{sum} [m]** the sum of costs for the traversed euclidean distance of each robot, expressed in meters; for the sake of physical interpretability, since simulation time steps are arbitrary in that they do not relate to an absolute time clock in seconds.

Since NAMO literature differentiates transfer from transit path components, we keep this differentiation and also compute the Sum of Transfer Distances $L_{transfer}$. We will also differentiate wait steps for the time-related criteria as to better understand how much time robots "waste" waiting to execute their plans.

We also keep track of several other relevant counters (which are summed across robots):

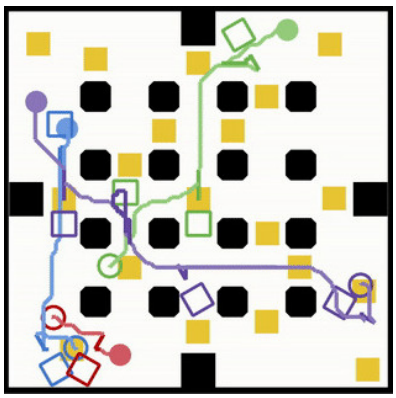
- **the number of postponements and replannings** (the initial plan for a goal is not counted as a replanning), so that coordination efficiency can be compared regardless of the time cost of actions in terms of steps,
- **the number of obstacle transfers**, since each obstacle grab would carry some risk of failure in itself in the real world and should happen as little as possible.
- **the number of successfully reached goals**, since as covered at the end of the experiments in Chapter IV, and reminded in the discussion of our coordination strategy, the robots may fail to meet some goals. While previously, failure could only be caused by the incapacity of the NAMO algorithm to solve the NAMO problem, it may now also occur because of the exhaustion of the maximum number of authorized planner calls.

As in Chapter IV we still measure the plan computation time (in seconds) as a measure of computation cost efficiency. In the result tables discussed in the next section, each criterion is averaged over the 200 scenarios of each experiment set presented in the previous section, and provided with the standard deviation.

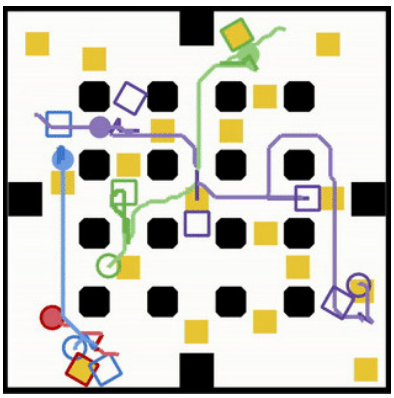
V. 3.3 Results

Each of the 5 sets of 200 scenarios is run using our coordination algorithm, with and without using our social placement cost model for choosing obstacle placements, respectively referred to as Coordinated NAMO (abb. C-NAMO) and Social Coordinated NAMO (abb. C-NAMO). Robot-dependent criteria are grouped in Table V.3, while world-dependent ones are grouped in table V.2. For each set of scenarios, the presented values are averages over the 200 scenarios of the set, with corresponding standard deviation.

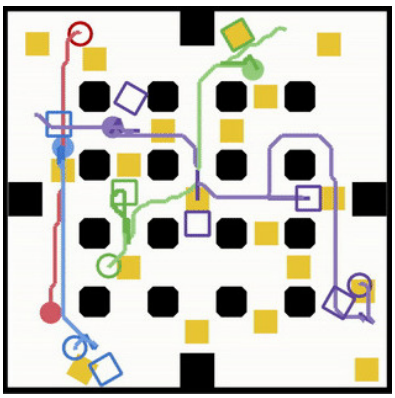
Plan improvement through Social & Coordinated NAMO (abb. SC-NAMO) The most relevant observation is that our social cost model reduces the number of postponements and replans by about 40% to 10% in the 'Intersections' scenario as the number of robots increases, as



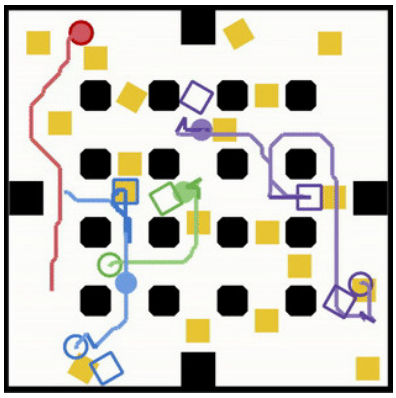
(A) First plan for all robots (after 1 step)



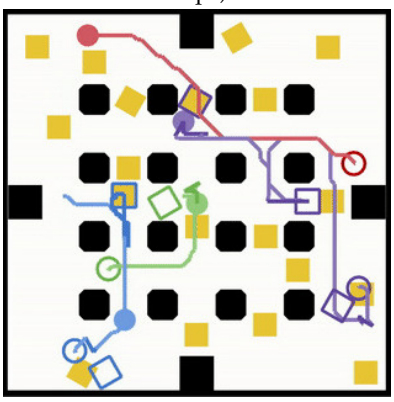
(B) Red reaches its first goal, blue and purple replanned (after 38 steps)



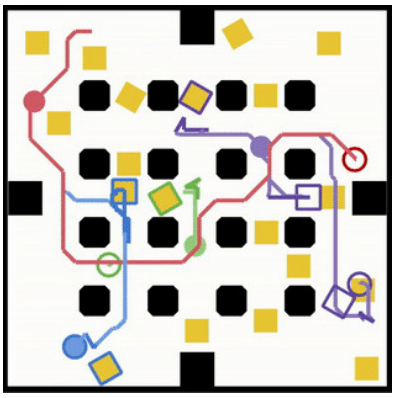
(C) Red plans to its second goal (after 41 steps)



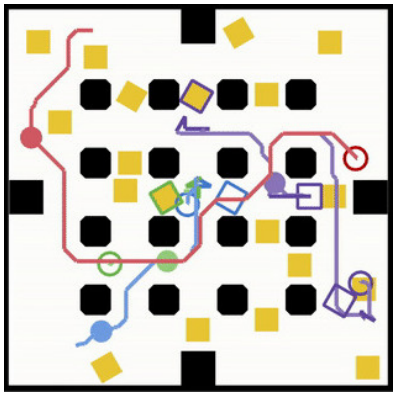
(D) Red reaches its second goal, purple, green and blue replanned (after 135 steps)



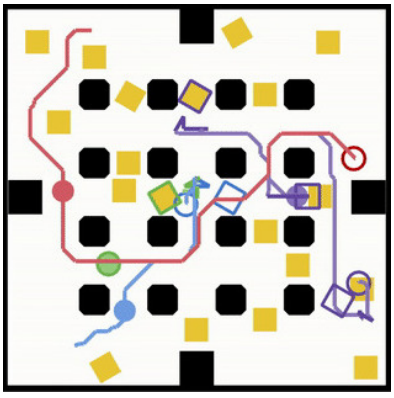
(E) Red plans to its third goal (after 144 steps)



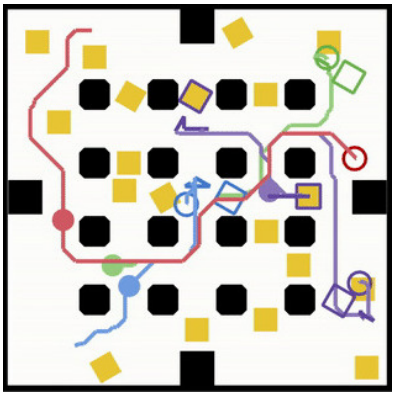
(F) Blue reaches its first goal, red and green replanned (after 170 steps)



(G) Blue planned to its second goal (after 182 steps)

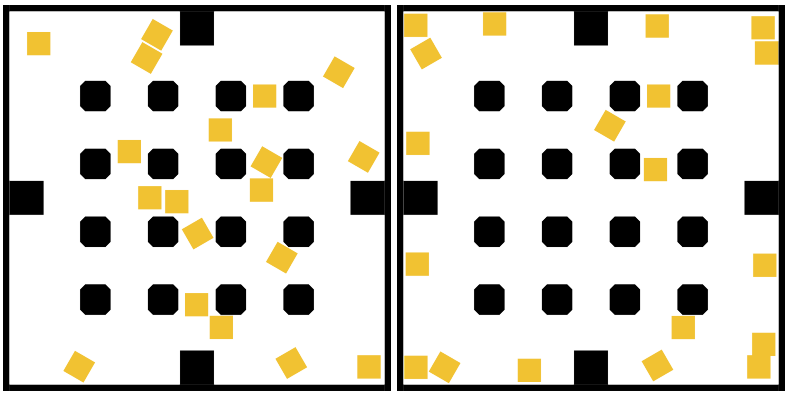


(H) Green reaches its first goal (after 197 steps)



(I) Green plans to its second goal (after 204 steps)

FIGURE V.21: Several snapshots of the simulation of the 4-robot scenario presented in Fig.V.20, assuming the use of Coordinated NAMO **without using our social occupation model**.



(A) Coordinated NAMO (abb. C-NAMO)

(B) Social Coordinated NAMO (abb. SC-NAMO)

FIGURE V.22: Resulting obstacle placements after the simulation of the 4-robot scenario presented in Fig.V.20 without using our social placement cost model (A) and with it (B).

can be seen in Table V.3. This results in a slightly lower number of wait steps, but that alone does not explain the significant drop in total number of steps to execute the goals (up to a fifth of the baseline). This drop can rather be explained by the reduction in total distance traveled L_{sum} as obstacles are mostly cleared from the intersections, resulting in significantly shorter transit paths. Fig.V.21 shows how, without social awareness, robots executing NAMO algorithms can quickly and badly affect one another's plans, as highlighted by the red robot's plan elongation between subfigures E and F. Though our social-awareness may come at the cost of almost up to 3 times longer transfer distances $L_{transfer}$, the number of transferred obstacles is reduced by a third or quarter in SC-NAMO. As stated in Section IV. 7.3.2, we remind that since Stilman's algorithm does not seek optimality, once most components are connected after a few goals, it only moves obstacles when goals are under them. For the CITI case, the ring configuration of the environment in between the rooms explains why results are not as outstanding: intersections are too far from each other.

Stable planning time As we had hoped, the $T_{planning}$ column in Table V.3 shows that opting for an implicit coordination approach prevented exponential explosion of planning time. SC-NAMO is, as previously observed in our mono-robot experiments (Cf. Section IV. 7.3.2) between 2 and 3 times more costly in terms of computation time due to the greater exploration of the robot's action space during transfer planning, which could be alleviated by using more aggressive heuristics, lazy intersection verification. However, our social cost model did not compensate the impact of an implicit coordination strategy on the goal success rate, which is similar between C-NAMO and SC-NAMO, ranging from 86% of success with 2 agents to 60% with 10 agents.

World State rearrangement Table V.2 clearly shows that whatever the number of robots, using our social cost model will still converge to a similar better space rearrangement: the total social cost drops by a third, accessible cells are always significantly more numerous and the

number of components only slightly lower (since NAMO still must connect components to operate). This better rearrangement can still be qualitatively observed in Fig.V.22.

Detailed and interactive graphs showing the aggregated criteria's evolution over time are available on our git repository (`https://gitlab.inria.fr/brenault/s-namo-sim`), with raw statistical data and end state snapshots for all runs.

Scenario Alg.	Number of components	Accessible cells	Social Cost
Initial conditions $W^{t_{init}}$	9	2606	636
Int - 2 $W^{t_{end}}$			
C-NAMO	2.1 ± 1.1	2549 ± 86	636 ± 22
SC-NAMO	$\mathbf{1.4} \pm 0.8$	$\mathbf{2800} \pm 73$	$\mathbf{402} \pm 47$
Int - 4 $W^{t_{end}}$			
C-NAMO	2.0 ± 1.0	2534 ± 86	631 ± 24
SC-NAMO	$\mathbf{1.4} \pm 0.6$	$\mathbf{2785} \pm 91$	$\mathbf{424} \pm 41$
Int - 5 $W^{t_{end}}$			
C-NAMO	2.0 ± 1.0	2548 ± 76	633 ± 26
SC-NAMO	$\mathbf{1.4} \pm 0.8$	$\mathbf{2774} \pm 91$	$\mathbf{434} \pm 46$
Int - 10 $W^{t_{end}}$			
C-NAMO	1.8 ± 0.9	2545 ± 89	639 ± 34
SC-NAMO	$\mathbf{1.6} \pm 0.8$	$\mathbf{2761} \pm 96$	$\mathbf{464} \pm 44$
Initial conditions $W^{t_{init}}$	40	7078	951
CITI - 2 $W^{t_{end}}$			
C-NAMO	28.8 ± 5.2	7018 ± 81	929 ± 23
SC-NAMO	$\mathbf{26.6} \pm 6.8$	$\mathbf{7209} \pm 94$	$\mathbf{760} \pm 99$

TABLE V.2: C-NAMO vs. SC-NAMO: World-dependent performance criteria comparison table, at world initial and end states. Values are averaged across 200 runs and provided with standard deviation. Number of robots is indicated in the first column with scenario name.

Scenario	Postponements	Replans	Wait Steps	Makespan t_{max} (total steps)	Successes / Goals	$L_{transfer}$ (m)	L_{sum} (m)	Transfers	$T_{planning}$ (s)
Int - 2									
C-NAMO	38 ± 61	27 ± 55	337 ± 672	5061 ± 875	43 ± 8 / 50	12.9 ± 4.0	425.2 ± 93.0	24 ± 6	83.4 ± 141.0
SC-NAMO	**24** ± 51	**16** ± 46	215 ± 561	**4196** ± 727	43 ± 8 / 50	31.9 ± 11.0	**359.2** ± 72.6	**16** ± 4	170.1 ± 336.6
Int - 4									
C-NAMO	47 ± 27	30 ± 24	400 ± 281	2940 ± 686	21 ± 4 / 25	7.3 ± 2.9	228.6 ± 57.2	13 ± 4	65.3 ± 117.7
SC-NAMO	**36** ± 28	**24** ± 25	332 ± 303	**2415** ± 635	20 ± 5 / 25	16.1 ± 7.1	**188.2** ± 52.5	**8** ± 3	154.0 ± 316.6
Int - 5									
C-NAMO	50 ± 24	32 ± 19	430 ± 250	2550 ± 740	16 ± 3 / 20	5.6 ± 2.4	191.0 ± 59.1	10 ± 3	60.5 ± 157.3
SC-NAMO	**39** ± 25	**25** ± 21	351 ± 263	**2065** ± 571	16 ± 4 / 20	13.4 ± 6.3	**154.8** ± 43.8	**7** ± 3	150.6 ± 204.0
Int - 10									
C-NAMO	61 ± 22	40 ± 17	535 ± 223	1479 ± 480	6 ± 2 / 10	2.6 ± 1.7	84.5 ± 32.8	4 ± 2	46.6 ± 53.8
SC-NAMO	**51** ± 22	**35** ± 17	475 ± 224	**1302** ± 422	6 ± 2 / 10	6.1 ± 4.3	**74.3** ± 28.2	**3** ± 2	155.8 ± 187.8
CITI - 2									
C-NAMO	20 ± 30	17 ± 27	220 ± 363	4985 ± 3980	23 ± 17 / 50	17.8 ± 14.4	434.2 ± 349.4	25 ± 20	205.2 ± 191.2
SC-NAMO	19 ± 30	14 ± 26	194 ± 360	5799 ± 4543	25 ± 19 / 50	43.5 ± 35.7	521.5 ± 411.7	13 ± 10	541.2 ± 480.2

TABLE V.3: C-NAMO vs. SC-NAMO: Agent-dependent performance criteria comparison table, cumulated over time. Values are averaged across 200 runs and provided with standard deviation. Unless specified otherwise in the first row, the unit is the number of occurrences. Number of robots is indicated in the first column with scenario name. Number of goals per robot is indicated in the 'Successes/Goals' column.

V. 4 Conclusions

This chapter first introduced the **new problem of General Multi-Robot Navigation Among Movable Obstacles**. We selected relevant additional hypotheses from the Multi-Robot Coordination literature (Cf. Section II. 2.2) to facilitate our exploration, and devised an **implicit coordination algorithm**. This algorithm has been designed to **allow concurrent use of existing NAMO algorithms** as is, while keeping the no-collision guarantee, **without requiring explicit communication**. **We reused our social occupation cost model** previously defined in Chapter IV **as a heuristic to help solve the fundamental sub-problem of deadlocks**, that arises in distributed multi-robot systems, which is especially harder to do when explicit communication is not allowed.

Experimental results on scenarios of increasing complexity with growing number of robots have shown that in a multi-robot setting, the **space rearranging properties of our social occupation cost model remain**, resulting in an overall improved navigation for all robots in the long term in close-intersections environments (lower traversed distance, waiting time and replanning calls). In the absence of explicit communication, our fully-implicit coordination strategy scales well computation-timewise at the cost of a decreasing success-rate, as the number of robots grows.

It is to be noted that, to the best of our knowledge, we are the first to ever explore Multi-Robot NAMO, setting a baseline for future work to compare against. We have opened the way, but our model could certainly be improved, again in particular in regard to its parametrization (postponement time, maximum umber of replans, Robot-Robot conflict detection horizon). Some variations to the proposed coordination algorithms could be studied, for instance not abandoning the current goal when no NAMO plan could be found when ignoring all obstacles. Indeed, counting on other robots to improve the environment arrangement, so that the waiting robot could try planning again later, when the NAMO problem became simpler, could have possibly improved the goal success rate too. Also, maybe not assuming the transported obstacle should be released before evading a deadlock could also yield interesting results. Finally, our additional study hypotheses could be relaxed to open an entirely new realm of usable algorithms and passionating experimentations, with heterogeneous, non-omniscient and explicitly communicating robots. **Our open simulation tool, algorithms and data provide a solid base to dive deeper in the study of this new problem**.

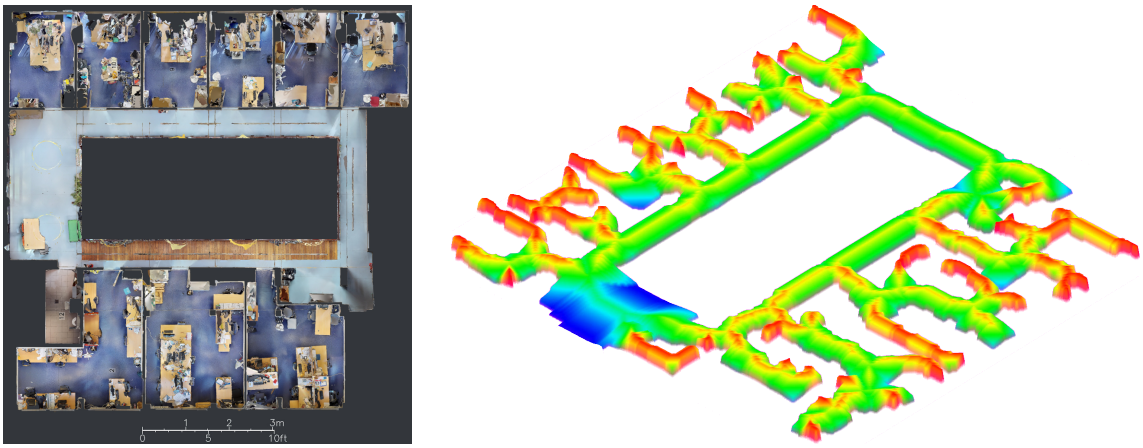
Chapter VI

Conclusion

In this thesis, we have made the first steps in bringing together the Navigation Among Movable Obstacles (abb. NAMO) Problem with the larger domains of Social Robotics and Multi-Robot Coordination. This marks a milestone in the process of bringing NAMO closer to real-world applicability in human environments.

We lowered the barrier of entry into NAMO research, with our extensive NAMO state of the art (Cf. Chapter II), open source simulation tool and datasets, all made from scratch. This, with our revision and implementation of two fundamental NAMO algorithms, (Wu&Levihn, 2014) and (Stilman, 2005), provides a stepping stone for reproducible and comparable NAMO experiments, with a common formalism (Cf. Chapter III). Our choice of standard, human-readable and well-supported formats such as SVG and JSON, for sharing our scenario datasets and output data, should also help with reproducibility and comparability. As of today, we have seen our implementation of (Wu&Levihn, 2014)'s algorithm already been reused by other scientists (e.g. [157]).

Our state of the art showed how, until now, that the only existing obstacle placement heuristic in NAMO was to minimize the robot's displacement cost, regardless of its impact on the environment's accessibility and navigability. We thus introduced and defined the problem of Socially-Aware NAMO, and proposed a social occupation cost model, based on two heuristic hypotheses as to the significance of free space in human environments: avoiding narrow spaces, and not leaving objects in the middle of space (Cf. Chapter IV). This model relies solely on the analysis of the environment's binary occupancy grid of fixed obstacles; a very low requirement both in terms of prior knowledge and computations, as this analysis only needs to be done once for a given environment. Using this model, we showed how to extend NAMO to S-NAMO algorithms that can compute plans resulting in more connected, less fragmented, and overall better arranged environments; thus improving social acceptability. Experimental results on scenarios of increasing complexity showed the scaling and generalization capabilities of our method; from a simple scenario with a single corridor and room, to a huge room with multiple crossings with many obstacles, to a real-world scan of our laboratory (Cf. Fig.VI.1). To the best of our knowledge, this is the first approach to propose an answer to social concerns in NAMO problems, and is the first study of the long-term impact of the use of NAMO algorithms. This model has recently been reused by other scientists as well [158].



(A) (B)

FIGURE VI.1: CITI Laboratory second floor. (A) Top-down view of the laboratory's 3D scan. (B) 3D-view of the social occupation costmap produced by our model. Blue costs are lower costs, red costs are high costs.

Finally, we defined the problem of Multi-Robot NAMO, and provided an implicit coordination algorithm that allows the concurrent use of existing NAMO algorithms in a Multi-Robot System while keeping the absence of collision guarantee - without explicit communication (Cf. Chapter V). Not requiring explicit communication for this first approach ensures maximum robustness regarding the possibility of individual robot or communication failure. We have also shown how our social occupation cost model can be used to decide suitable placement of robots in the same way as movable obstacles, providing a relevant heuristic to solve a number of deadlock situations. This coordination algorithm allowed us to verify the preservation of positive environment rearrangement properties of our social occupation cost model observed in mono-robot NAMO experiments. More importantly, we have witnessed the appearance of positive effects of this rearrangement, reducing parasite interactions between the robots that require replanning or waiting. Again, to the best of our knowledge, this is the first approach to propose an answer to multi-robot concerns in NAMO problems, which will provide a baseline for future works to build upon or compare against.

VI. 1 Perspectives

In a way, one of the main contributions of this thesis has been to introduce the two open problems of Socially-Aware and Multi-Robot NAMO. NAMO itself still being an open problem, we could only but scratch the surface of their multiple implications, and much remains to be explored.

Consolidating existing NAMO research While we extensively studied the current NAMO state of the art in Chapter II, we could only study in details and implement the two most influential existing algorithms (Cf. Chapter III). Implementing the rest of these algorithms, eventually leading to an actual robust computer library people can pick their preferred algorithm

from, remains a very relevant challenge today. Of course, the logical corollary would be to augment our current dataset, with new NAMO environments and scenarios that would best showcase these others algorithms' advantages and drawbacks. Finally, benchmarking said algorithms with said augmented dataset within a same experimental context (i.e. same machines with same characteristics) would also help in choosing which algorithm to use depending on context.

Incorporating human feedback and knowledge As mentioned in the conclusion of Chapter IV, improving our social occupation cost model would mainly require larger and more diverse human input, in order to tune its parametrization and validate its social acceptability with actual human feedback. Setting up appropriate experiments requiring human participation would also require the input of social sciences experts, that would ensure their adherence to established experiment design guidelines. While the potential extension of our model with any of the other models presented in Section II. 2 or IV. 3 would be technically straightforward thanks to the commonplace nature of our grid-based approach, the appropriate choice of weight parameters and validation would also require the input of more humans and social sciences experts. While non-trivial, the resulting model would certainly yield more meaningful and relevant obstacle placements (and maybe robot placements in MR-NAMO deadlock situations).

Relaxing our additional Multi-Robot hypotheses The ultimate conclusion of Chapter V was that there is a lot of interesting new research to be done by simply relaxing some, or all of our additional Multi-Robot work hypotheses presented in Section V. 1.2. We thus identified the following four main exploration axes:

- **Heterogeneous Multi-Robot System:** Allowing heterogeneous robots with varying action spaces would for instance allow the exploration of new cooperative algorithms, that would encourage stronger robots to open the way for weaker robots by moving objects they can't manipulate, even if these objects don't impede the strong robot's path. Conversely, small robots could use our social occupation cost model to open a larger way than they need to pass, in prevision for larger robots to pass later.
- **Explicit Communicating Multi-Robot System:** Allowing robots to explicitly communicate about their plans would certainly further improve such cooperative behaviors. It would also provide significant improvements and guarantees in terms of problem resolution capabilities, compared to any fully implicit approach (e.g. deadlock resolution guarantee, resolution completeness, ...). This would hold more or less true depending on the exact type of coordination (centralized or decentralized, local or global, ...). Similarly, communicating their sensing data/maps would also facilitate NAMO planning under limited sensing constraints (in opposition to an omniscient view of the world). It would most notably allow all robots to keep track of both other robots and their manipulated obstacles, resulting in fewer replanning due to bad estimates of the world state.

- **Many-to-one robot-obstacle relationship:** If the one-robot-per-obstacle hypothesis were relaxed, it could allow the coordinated manipulation of hard-to-move obstacles that a single robot could not move alone, be it because of geometry, or more elaborate kinodynamic characteristics. Of course, assuming explicit communication about plans or sensing data would facilitate the development of such coordination.
- **Human Agents:** One could even go as far as imagining the collaborative resolution of a NAMO problem between robots and humans. This study could go both ways: robots helping humans navigate among obstacles, and vice versa. In both cases, explicit communication (e.g. through vocal recognition and speech synthesis) would likely be beneficial in achieving better planning outcomes.

VI. 2 Beyond planning, experimenting in the real world

Beyond our contributions, and the research perspectives mentioned above, *there is much to be done outside path planning* for NAMO research to enter our daily lives. Applied NAMO research needs to move beyond controlled laboratory environments. For this, much work still needs to be done in terms of sensing, mapping, and tracking of movable obstacles to allow the application of existing algorithms in a useful and trustworthy manner. In the real world, robots need to accurately evaluate the semantics, geometry and kinodynamics of obstacles in a coherent manner over time, as to provide a trustworthy-enough world model that NAMO planners can rely upon to generate successful NAMO plans. To this end, and in order to bring this thesis' contributions closer to real-world application, a research engineer has been recruited by INRIA. We hope this work contributes in kindling renewed interest in NAMO, and may serve new advances in the field, as we believe that autonomously manipulating obstacles is an essential skill for robots to properly operate in human environments.

Appendix A

A* Algorithm

Algorithm 12: Generic A* Algorithm

Data: Start configuration *start*, Goal configuration *goal*, graph dependent subfunctions EXIT-CONDITION, GET-NEIGHBORS, HEURISTIC, COST

Result: Whether a path was found, last evaluated configuration *current*, precedence dictionary *cameFrom*, set of evaluated configurations *closeSet*, dictionary of scores *gscore*, queue of configurations to evaluate *openQueue*;

```
1  cameFrom ← empty map;
2  current ← NULL;
3  openQueue ← MAKE-PRIORITY-QUEUE(HEURISTIC(start, goal), start);
4  closedSet ← ∅;
5  gscore ← {start : 0};
6  while openQueue ≠ ∅ do
7      current ← POP(openQueue);
8      if EXIT-CONDITION(current, goal) then
9          return True, current, cameFrom, closeSet, gscore, openQueue;
10     if current ∈ closeSet then continue;
11     else add current to closeSet;
12     for neighbor in GET-NEIGHBORS(current) do
13         tentativeGscore ← gscore[current]+ C(current, neighbor);
14         if neighbor ∉ gscore or tentativeGscore < gscore[neighbor] then
15             cameFrom[neighbor] ← current;
16             gscore[neighbor] ← tentativeGscore;
17             PUSH(openQueue, (tentativeGscore + HEURISTIC(neighbor, goal), neighbor));
18 return False, current, cameFrom, closeSet, gscore, openQueue;
```

Appendix B

Original (Wu&Levihn, 2014)'s Algorithm

For the sake of readability, end of bloc statements (e.g. **end if**) were removed in favor of lined indentations - otherwise, the code remains completely unchanged from the original [34]. The algorithms are meant to be read successively - the associated tables only describe the variables, operators and functions once.

Variable	Description
$R\ q_r^t$	Current robot configuration
q_r^s	Start robot configuration
q_r^g	Goal robot configuration
$\mathcal{O}_{new}$	All newly detected or updated obstacles since the last plan computation
$minCost$	Heuristic list of obstacles, ordered by increasing cost to move the obstacle + cost to go from the obstacle to the goal after move
$euclidianCost$	Heuristic list of obstacles, ordered by increasing euclidean distance from the obstacle to the goal
mC_pt	Traversal index for $minCost$, always points to the next best unevaluated obstacle manipulation partial plan in the list
eC_pt	Traversal index for $euclidianCost$, always points to the next best unevaluated obstacle manipulation partial plan in the list
mC	Variable from GET_NEXT function, indicates if the returned partial $\mathcal{P}_{next}$ plan is associated with an obstacle from $minCost$ or not
M_i	Currently evaluated obstacle
$\mathcal{P}^*$	Current best Plan
$\mathcal{P}_{next}$	Next best partial plan to the goal
$\mathcal{P}$	Full plan for currently evaluated obstacle M_i

TABLE B.1: Variables table for Algorithm 13

Operator/Function	Description
$\Leftarrow =$	Assignment operator (= also used for equality check)
A^*	A* search Algorithm described in Appendix A
$\cup$	Set update union
$\cap$	Geometric intersection
$\in$	Element of
$\notin$	Not element of
$\mathcal{S}$	Swept area of a plan
$\mathcal{C}$	Cost of a plan
[]	Create tuple
$Cost_l$	Heuristic cost used by *minCost* (move the obstacle + move from the obstacle to goal)
$\equiv A_M$	Check if step is a manipulation action
GET-NEW-INFORMATION	Returns the list of newly detected or updated obstacles
UPDATE	Inserts or modifies the element in the heuristic list, respecting the list's order

TABLE B.2: Operators/Functions table for Algorithm 13

Variable	Description
$\mathcal{P}_{mC}$	Next best partial plan to the goal
$\mathcal{P}_{eC}$	Full plan for currently evaluated obstacle M_i

TABLE B.3: Variables table for Algorithm 14

Operator/Function	Description
$++$	Unit increment operator

TABLE B.4: Operators/Functions table for Algorithm 14

Variable	Description
d	Unit movement vector in an axis-aligned direction in $\{left, right, up, down\}$
$q_{M_i}^{tmp}$	Currently evaluated Robot configuration while moving obstacle M_i
$q_{M_i}^{t}$	Robot configuration at the obstacle's side before move
$closedList$	List of fully evaluated configurations
$\mathcal{Q}$	Queue of configurations to evaluate
$\mathcal{P}_{to\_obstacle}$	Path to reach the obstacle
$\mathcal{P}_x$	Path to move the obstacle
$\mathcal{P}_{to\_goal}$	Path from the obstacle to the goal
$\mathcal{P}_{M_i}^*$	Next best partial plan to the goal
$\mathcal{P}$	Currently evaluated full plan

TABLE B.5: Variables table for Algorithm 15

Operator/Function	Description
$\mathcal{C}_l$	Equivalent to $Cost_l$ described in table B.2
$\mathcal{C}_{est}$	Estimation of the current estimated plan cost (cost to obstacle + cost of manipulation so far + euclidean distance to goal cost estimate)
is valid	path found check
new opening was created	Local opening detection algorithm call (see Appendix D)
APPEND	Add unit movement vector to plan
HAS_ELEMENT	element of set check
GET_POSITION	get robot configuration at the currently evaluated plan step
POP	returns queue's next element
!	Would be logical NOT, but is actually a typo at line 14 (the robot would only add fully evaluated nodes to the queue, which is the opposite of what is needed).
skip	immediately start next loop

TABLE B.6: Operators/Functions table for Algorithm 15

Algorithm 13: Original (Wu&Levihn, 2014) main robot control routine

Data: Robot start configuration q_r^s, Robot goal configuration q_r^g

```
1  OPTIMIZED(q_r^s, q_r^g)
2      R ⇐ q_r^s
3      minCost, euclidianCost ⇐ ∅
4      mC_pt, eC_pt = 0
5      P* ⇐ A*(q_r^s, q_r^g)
6      while R ≠ q_r^g do
7          O_new ⇐ O_new ∪ GET-NEW-INFORMATION()
8          if S(P*) ∩ O_new ≠ ∅ then
9              P* ⇐ A*(q_r^t, q_r^g)
10             for each M_i ∈ O_new do
11                 UPDATE(euclidianCost, M_i)
12             P_next = GET-NEXT(mC_pt, eC_pt)
13             while C(P*) ≥ C(P_next) do
14                 if mC = true or P_next ∉ minCost then
15                     P = OPT-EVALUATE-ACTION(P_next, P*)
16                     UPDATE(minCost, [M_i, Cost_l(M_i)])
17                     if C(P) < C(P*) then
18                         P* = P
19                     P_next = GET-NEXT(mC_pt, eC_pt)
20             O_new ⇐ ∅
21         mC_pt, eC_pt = 0
22         if Next step in P* ≡ A_M then
23             minCost ⇐ ∅
24         R ⇐ Next step in P*
```

Algorithm 14: Original (Wu&Levihn, 2014) obstacle selection routine

Data: Respective obstacle heuristic lists ($minCost, euclidianCost$) traversal indexes mC_pt, eC_pt

Result: indicator mC (of whether the returned partial plan is associated with an obstacle from $minCost$ or not), Next best partial plan to the goal $\mathcal{P}_{next}$

```
1  GET-NEXT(mC_pt, eC_pt)
2      mC = false
3      P_mC = minCost(mC_p)
4      P_eC = euclidianCost(eC_p)
5      if C(P_mC) ≤ C(P_eC) then
6          P_next = P_mC
7          mC_pt++
8          mC = true
9      else
10         P_next = P_eC
11         eC_pt++
12         mC = false
13     return (mC, P_next)
```

Algorithm 15: Original (Wu&Levihn, 2014) 3-components path planning routine

Data: Obstacle to evaluate for movement M_i, current best plan $\mathcal{P}^*$

Result: Next best partial plan to the goal $\mathcal{P}^*_{M_i}$, associated heuristic cost $\mathcal{C}_l(M_i)$

```
1  OPT-EVALUATE-ACTION(M_i, P*)
2      P*_{M_i}, Q, closedList ⇐ ∅
3      C_l(M_i) ← ∞
4      P_{to_obstacle} = A*(q_r^t, q_{M_i}^t)
5      if P_{to_obstacle} NOT valid then
6          return failure
7      C(P*_{M_i}) = ∞
8      INSERT(Q, P*_{M_i}))
9      INSERT(closedList, q_{M_i}^s)
10     while Q ≠ ∅ do
11         P_x ⇐ POP(Q)
12         for d in {left, right, up, down} do
13             q_{M_i}^{tmp} ⇐ GET_POSITION(M_i, P_x) + d
14             if !HAS_ELEMENT(closedList, q_{M_i}^{tmp}) then
15                 skip
16             INSERT(closedList, q_{M_i}^{tmp})
17             if q_{M_i}^{tmp} causing robot or obstacle collision then
18                 skip
19             APPEND(P_x, d)
20             if C_est(P_x) ≥ C(P*_{M_i}) or C_est(P_x) ≥ C(P*) then
21                 skip
22             if new opening was created then
23                 P_{to_goal} = A*(q_{M_i}^{tmp}, q_r^g)
24                 if P_{to_goal} is valid then
25                     P = P_{to_obstacle} + P_x + P_{to_goal}
26                     if C(P_x) + C(P_{to_goal}) < C_l(M_i) then
27                         C_l(M_i) ← C(P_x) + C(P_{to_goal})
28                     if C(P) < C(P*_{M_i}) then
29                         P*_{M_i} = P
30             INSERT(Q, P_x)
31     if P*_{M_i} was set then
32         return (P*_{M_i}, C_l(M_i))
33     else
34         return ∅
```

Appendix C

Original (Stilman, 2005)'s Algorithm

The code remains completely unchanged from [19].

Variable	Description
W^t	Physical world representation at the planning time t
W^{t+2}	Physical world representation after navigation to an obstacle (+1) and manipulation of said obstacle(+2)
r^f	Goal robot configuration
x^f	Goal robot cell
$PrevList$	Best-First Search list of evaluated (obstacle, free space component) pairs
$AvoidList$	Optional list keeping track of the free space components traversed in the search branch - prevents the heuristic RCH search from returning in them
C_R^{acc}	Set of accessible cells for the robot from its current configuration / Current connected component of the free space of the robot
τ_M	Manipulation/Transfer path (sequence of robot configurations while holding the obstacle at a chosen grasping point)
τ_N	Navigation/Transit path (sequence of robot configurations without holding an obstacle)
$FuturePlan$	List of all subsequent transit and transfer paths required to reach the goal, coming from the deeper recursions
c	Cost of the Manipulation/Transfer path
$\emptyset$	Empty set
NIL	Empty path / plan
O_1	Obstacle to evaluate movement for, in order to connect the current robot accessible space with another free space component
C_1	Free space component to connect to the current robot accessible space in the current obstacle manipulation evaluation

TABLE C.1: Variables table for Algorithm 16

Algorithm 16: Original (Stilman, 2005) Obstacle Choice heuristic.

Data: Current world state W^t, free space components to avoid *PrevList*, goal configuration r^f
Result: A valid NAMO Plan P or NIL

```
1  SELECT-CONNECT(W^t, PrevList, r^f)
2      AvoidList ← ∅
3      if x^f ∈ C_R^acc(W^t) then
4          return (FIND-PATH(W^t, x^f))
5      while (O_1, C_1) ← RCH(W^t, AvoidList, PrevList, r^f) ≠ NIL do
6          (W^{t+2}, τ_M, c) ← MANIP-SEARCH(W^t, O_1, C_1)
7          if τ_M ≠ NIL then
8              FuturePlan ← SELECT-CONNECT(W^{t+2}, PrevList append C_1, r^f)
9              if FuturePlan ≠ NIL then
10                 τ_N ← FIND-PATH(W^t, τ_M[0])
11                 return ((τ_N, τ_M) append FuturePlan)
12         AvoidList append (O_1, C_1)
13     return NIL
```

Algorithm 17: Original (Stilman, 2005) Obstacle Choice heuristic

Data: Current world state W^t, (O_i, C_i) pairs to avoid *AvoidList*, free space components to avoid *PrevList*, goal configuration r^f
Result: Relevant obstacle to consider for manipulation O_1, initial free space component C_1 to be joined with current accessible cells C_R^{acc}

```
1  RCH(W^t, AvoidList, PrevList, r^f)
2      Closed ← ∅
3      Q ← MAKE-PRIORITY-QUEUE(r^t, 0, 0)
4      while Q ≠ empty do
5          x_1 = (r_1, O_F, C_F) = REMOVE-FIRST(Q)
6          if x_1 ∈ Closed then continue
7          if (r_1 = r^f and O_F ≠ 0 and C_F ≠ 0) then return (O_F, C_F)
8          Closed append (x_1)
9          for each r_2 ∈ ADJACENT(r_1) do
10             if (C_F ≠ 0) then ENQUEUE(Q, (r_2, O_F, C_F)); continue
11             if (O_F ≠ 0 and r_2 ∈ C_R^free) then ENQUEUE(Q, (r_2, O_F, 0))
12             if (O_F ≠ 0 and r_2 exc χ_R^{O_F}) then ENQUEUE(Q, (r_2, O_F, 0))
13             if (O_F ≠ 0 and r_2 ∈ C_i s.t. C_i ∉ PrevList and (O_F, C_i) ∉ AvoidList) then
14                 ENQUEUE(Q, (r_2, O_F, C_i))
15             if (O_F = 0 and r_2 ∈ C_R^free) then ENQUEUE(Q, (r_2, 0, 0))
16             if (O_F = 0 and r_2 ∈ exc χ_R^{O_i}) then ENQUEUE(Q, (r_2, O_i, 0))
17     return NIL
```

Operator/Function	Description
gets	Assignation operator
$\in$	Element of
FIND-PATH	Navigation/Transit path planner call (A* call on the discretized grid of W^t)
MANIP-SEARCH	Manipulation/Transfer path planner call (Breadth-First search call assuming uniform action costs)
append	Inserts elements from the right list at the end of the left list
[]	Get list element at index between

TABLE C.2: Operators/Functions table for Algorithm 16

Variable	Description
Closed	Set of fully evaluated configurations
$\emptyset$ **empty**	Empty set
Q	Priority queue of configurations to evaluate (ordered by increasing cost + heuristic distance cost to goal (e.g. euclidean distance))
r^t	Robot start configuration at the function's call
x_1	Currently evaluated state (r_1, O_F, C_F)
r_1	Current robot configuration (e.g. currently evaluated grid cell)
r_2	Neighbor robot configuration from r_1 (accessible with a single action)
O_F	First traversed obstacle in path to cell (0 if no traversed obstacle yet)
C_F	First traversed component in path to cell (0 if no traversed component other than current robot component yet)
O_i	Obstacle exclusively colliding with the robot at configuration r_2
C_i	Free space at robot configuration r_2
NIL	Failure to find a relevant obstacle/free space component to connect

TABLE C.3: Variables table for Algorithm 17

Operator/Function	Description
MAKE-PRIORITY-QUEUE	Returns a new priority queue with its first element
$= \neq$	Inequality check
REMOVE-FIRST	Pop first (lowest cost) element in queue
$\in \notin$	(Not) Element of
continue	Immediately start next loop
ADJACENT	Returns the set of all accessible robot configurations from the current one r_1
C_R^{free}	Set of all collision-free robot configurations (union of all free space components)
ENQUEUE	Inserts element in the priority queue
exc χ_R	Checks if robot configuration is exclusively colliding with a single obstacle
s.t.	So that

TABLE C.4: Operators/Functions table for Algorithm 17

The following algorithm is our own formalization of the obstacle transfer/manipulation subroutine described in [3, 19] (MANIP-SEARCH), derived from their textual statements.

Algorithm 18: (Interpreted) original (Stilman, 2005) obstacle manipulation/transfer search routine

Data: Current world state W^t, Relevant obstacle to consider for manipulation O_1, Connected Component C_1 to be joined with currently accessible cells C_R^{acc}

Result: Evaluated world state after manipulation W^{t+2}, Manipulation plan τ_M

1 `MANIP-SEARCH(`W^t, O_1, C_1`)`
2 $\quad \tau_M \leftarrow$ BREADTH-FS($(\emptyset,\emptyset,\emptyset)$, GET-NEIGHBORS, EXIT-CONDITION)
3 $\quad W^{t+2} \leftarrow$ COPY-AND-UPDATE(W^t, τ_M)
4 $\quad$ **return** W^{t+2}, τ_M
5 `GET-NEIGHBORS(`$(p, q_R^{manip}, q_R^{cur})$`)`
6 $\quad$ *neighbors* $\leftarrow \emptyset$
7 $\quad$ **if** $p, (q_R^{manip}, q_R^{cur}) = (\emptyset,\emptyset,\emptyset)$ **then**
8 $\quad\quad$ **for** $(p, q_R^{manip}) \in AReach(O_1)$ **do**
9 $\quad\quad\quad$ APPEND(*neighbors*, $(p, q_R^{manip}, q_R^{manip})$)
10 $\quad\quad$ **return** *neighbors*
11 $\quad$ **for** $A_M \in \mathcal{A}_M$ **do**
12 $\quad\quad$ $R^{next}, O_1^{next} \leftarrow$ APPLY-ACTION((R^{cur}, O_1^{cur}), A_M)
13 $\quad\quad$ **if** $(p, q_R^{manip}, q_R^{next}) \in closedList$ **then skip**
14 $\quad\quad$ **if** $R^{next} \cap W^t_{/O_1,R}$ **or** $O_1^{next} \cap W^t_{/O_1,R}$ **then skip**
15 $\quad\quad$ APPEND(*neighbors*, $(p, q_R^{manip}, q_R^{next})$)
16 $\quad$ **return** *neighbors*
17 `EXIT-CONDITION(`$(p, q_R^{manip}, q_R^{cur})$`)`
18 $\quad q_R^{C_1} \leftarrow q_R^g$
19 $\quad$ **if** $q_R^g \notin C_1$ **then** $q_R^{C_1} \leftarrow$ RANDOM-FREE-CONFIGURATION(W_t, C_1)
20 $\quad \mathcal{P}_{to\_C_1} \leftarrow A^*(q_R^{cur}, q_R^{C_1})$
21 $\quad$ **if** $\mathcal{P}_{to\_C_1} \neq \emptyset$ **then return** True
22 $\quad$ **return** False

Appendix D

Original Efficient Opening Detection Algorithm

Variables, operators and functions that do not change meaning depending on the algorithm are not relisted in the later tables.

Variable	Description
G	2D binary occupancy grid
M_i	The obstacle being moved
$\mathcal{A}_M$	The manipulation action applied on M_i
M	The subgrid of G encompassing the inflated footprint of M_i by the robot's diameter
$x\_offset\ y\_offset$	Subgrid Upper-left corner coordinates in the global grid G
BA	Initial state Blocking Areas matrix (shape of M, non-zero values where other obstacles in G intersect with the inflated footprint of M_i by the robot's diameter)
BA_s	Final state (after M_i is moved with action $\mathcal{A}_M$) Blocking Areas matrix
BA_s^*	Shifted BA_s matrix in the direction of $\mathcal{A}_M$ (e.g. if M_i is moved one cell to the right, BA_s's columns are shifted one column to the right (thus the last/rightmost column is dropped), and the first/leftmost column is filled with zeros)
$i\ j$	Traversal indexes of the shifted BA_s matrix
$x\ y$	Shifted coordinates in BA_s^* from BA_s
Z	Buffer variable to store the result of the COMPARE call

TABLE D.1: Variables table for Algorithm 19

Operator/Function	Description
$=$	Assignment and equality check operator
GET-M_i'-MATRIX	Returns the subgrid of G encompassing the inflated footprint of the given obstacle by the robot's diameter
$.x$ $.y$	Returns the subgrid Upper-left corner coordinates in the global grid G
GET-BLOCKING-AREAS	Returns a new matrix the shape of M, with non-zero values where other obstacles in G intersect with the inflated footprint of M_i by the robot's diameter
GET-NEW-X-POS GET-NEW-Y-POS	Returns the subgrid upper-left corner coordinates in the global grid G
[0][0] ; {0-Matrix; dim()=dim(M)}	Returns a zero matrix with the shape of M
$\rightarrow$	Returns the range of integers from left operand to right operand
$\|\|$	Returns the length of the operand
[]	Returns element at index
$\equiv$	Equality check for matrices

TABLE D.2: Operators/Functions table for Algorithm 19

Variable	Description
BA	Currently built Blocking Areas matrix (shape of M, non-zero values where other obstacles in G intersect with the inflated footprint of M_i by the robot's diameter)
x_off y_off	BA matrix upper-left corner coordinates in global grid G
$index$	Arbitrary unique identifier for each blocking area
x y	Traversal indexes of M

TABLE D.3: Variables table for Algorithm 20

Operator/Function	Description
$\neq$	Inequality check

TABLE D.4: Operators/Functions table for Algorithm 20

Variable	Description
BA	Same currently built Blocking Areas matrix as in Table D.3
x y	Coordinates in BA of the cell to set the $index$ for
$index$	Same index as in Table D.3
i j	Traversal indexes for the chessboard (3×3) neighborhood of the (x, y) cell

TABLE D.5: Variables table for Algorithm 21

Variable	Description
BA	Same Initial state Blocking Areas matrix as in Table D.1
BA_s^*	Same Final state Blocking Areas matrix as in Table D.1
del_num	Set of Indexes/Arbitrary identifiers of the blocking areas in BA that intersect with blocking areas in BA_s^*
$\emptyset$	Empty set
BS	Initially a copy of BA, should be the returned variable for the pseudocode to actually be valid
$x\ y$	Traversal indexes of the BS matrix

TABLE D.6: Variables table for Algorithm 22

Operator/Function	Description
$:=$	Assignment operator
$\in$	Element of
$\cup$	Set union update

TABLE D.7: Operators/Functions table for Algorithm 22

Algorithm 19: Original Efficient Opening Detection Algorithm

Data: Binary occupancy grid G, Manipulated obstacle M_i, Manipulation action $\mathcal{A}_M$
Result: True if new local opening detected, False otherwise

```
CHECK-NEW-OPENING(G, M_i, A_M)
    M = GET-M_i'-MATRIX(M_i)
    x_offset = M_i.x
    y_offset = M_i.y
    BA = GET-BLOCKING-AREAS(x_offset, y_offset)
    x_offset = GET-NEW-X-POS(A_m, M_i)
    y_offset = GET-NEW-Y-POS(A_m, M_i)
    BA_s = GET-BLOCKING-AREAS(x_offset, y_offset)
    BA_s^* = [0][0] ; {0-Matrix; dim()=dim(M)}
    for i = 0 → |BA_s^*| do
        for j = 0 → |BA_s^*[i]| do
            x = (x_offset − M_i.x) + i
            y = (y_offset − M_i.y) + j
            if 0 < x < |BA_s^*| AND 0 < y < |BA_s^*[x]| then
                BA_s^*[x][y] = BA_s[i][j]
    Z = COMPARE(BA, BA_s^*)
    if Z ≡ [0][0] then
        return false
    return true
```

Algorithm 20: Blocking Areas detection subroutine

Data: Origin coordinates of the inflated obstacle grid x_off, y_off
Result: Blocking areas BA

```
1 GET-BLOCKING-AREAS(x_off, y_off)
2     index = 1
3     BA = [0][0] ; {0-Matrix; dim()=dim(M)}
4     for x = 0 → |M| do
5         for y = 0 → |M[x]| do
6             if M[x][y] ≠ 0 AND G[x + x_off][y + y_off] ≠ 0 then
7                 ASSIGN-NR(BA, x, y, index)
8     return BA
```

Algorithm 21: Blocking Area index assignment and overwrite subroutine

Data: Blocking Areas matrix BA, Coordinates x, y of cell to assign the index to, Current Blocking Area $index$

```
1 ASSIGN-NR(BA, x, y, index)
2     for i = −1 → 1 do
3         for j = −1 → 1 do
4             if BA[x + i][y + j] ≠ 0 then
5                 BA[x][y] = BA[x + i][y + j]
6                 return
7     BA[x][y] = index
8     index = index + 1
9     return
```

Algorithm 22: Blocking Areas intersection computation routine

Data: Initial state Blocking Areas matrix BA, Final state Blocking Areas matrix BA_s^*
Data: Comparison matrix BA

```
1 COMPARE(BA, BA*_s)
2     del_num := ∅
3     for x = 0 → |BS| do
4         for y = 0 → |BS[x]| do
5             if BA[x][y] ∈ del_num then
6                 BS[x][y] = 0
7             if BA[x][y] ≠ 0 AND BA*_s[x][y] ≠ 0 then
8                 del_num = del_num ∪ BS[x][y]
9                 BS[x][y] = 0
10    return BA
```

Bibliography

[1] P. C. Chen and Y. K. Hwang, "Practical path planning among movable obstacles," Sandia National Labs., Albuquerque, NM (USA), Tech. Rep. SAND-90-2383C; CONF-910451-7, 1990.

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

[3] M. Stilman and J. J. Kuffner, "Navigation among movable obstacles: real-time reasoning in complex environments," *International Journal of Humanoid Robotics*, vol. 02, no. 4, pp. 479–503, 2005.

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

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

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

[7] 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*, Nov. 2014, pp. 86–91.

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

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

[10] 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*, May 2013, pp. 3832–3838.

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

[12] E. Mueggler, M. Faessler, F. Fontana, and D. Scaramuzza, "Aerial-guided navigation of a ground robot among movable obstacles," in *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014)*, Oct. 2014, pp. 1–8.

[13] J. Rios-Martinez, A. Spalanzani, and C. Laugier, "From Proxemics Theory to Socially-Aware Navigation: A Survey," *International Journal of Social Robotics*, vol. 7, no. 2, pp. 137–153, Apr. 2015.

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

[15] D. V. Lu, "Social navigation layers - ROS Wiki," available at: http://wiki.ros.org/social_navigation_layers, last edited: 2014-11-10, last viewed: 2024-01-25.

[16] J. K. Verma and V. Ranga, "Multi-Robot Coordination Analysis, Taxonomy, Challenges and Future Scope," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 1, p. 10, Apr. 2021.

[17] M. Bellusci, N. Basilico, and F. Amigoni, "Multi-Agent Path Finding in Configurable Environments," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, May 2020, pp. 159–167.

[18] D. Vainshtain and O. Salzman, "Multi-agent terraforming: Efficient multi-agent path finding via environment manipulation," *Proceedings of the International Symposium on Combinatorial Search*, vol. 12, no. 1, pp. 239–241, 2021.

[19] M. Stilman, "Navigation Among Movable Obstacles," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, Oct. 2007.

[20] F. Lindner and C. Eschenbach, "An Affordance-Based Conceptual Framework for Spatial Behavior of Social Robots," in *Sociality and Normativity for Robots*, R. Hakli and J. Seibt, Eds. Cham: Springer International Publishing, 2017, pp. 137–158.

[21] ——, "Affordance-Based Activity Placement in Human-Robot Shared Environments," in *Social Robotics Series Title: Lecture Notes in Computer Science*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz,

C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, G. Herrmann, M. J. Pearson, A. Lenz, P. Bremner, A. Spiers, and U. Leonards, Eds., vol. 8239. Cham: Springer International Publishing, 2013, pp. 94–103.

[22] R. Limosani, L. Y. Morales, J. Even, F. Ferreri, A. Watanabe, F. Cavallo, P. Dario, and N. Hagita, "Long-term human affordance maps," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept. 2015, pp. 5748–5754.

[23] Y. Jiang, H. S. Koppula, and A. Saxena, "Modeling 3D Environments through Hidden Human Context," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 10, pp. 2040–2053, Oct. 2016.

[24] O. Ben-Shahar and E. Rivlin, "Practical pushing planning for rearrangement tasks," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 4, pp. 549–565, Aug. 1998.

[25] International Federation of Robotics, "World Robotics 2021," International Federation of Robotics, Tech. Rep., 2021.

[26] E. Ackerman, "With New Roomba j7, iRobot Wants to Understand Our Homes," *IEEE Spectrum*, Sept. 2021.

[27] M. Levihn, "Autonomous environment manipulation to facilitate task completion," PhD Thesis, Georgia Institute of Technology, Atlanta, Georgia, Mar. 2015.

[28] J. Scholz, "Physics-based reinforcement learning for autonomous manipulation," Ph.D. dissertation, Georgia Institute of Technology, Aug. 2015.

[29] E. Ackerman, "iRobot Completely Redesigns Its Floor Care Robots With New m6 and s9," *IEEE Spectrum*, May 2019.

[30] Benoit Renault, Jacques Saraydaryan, and Olivier Simonin, "Towards S-NAMO: Socially-aware Navigation Among Movable Obstacles," in *RoboCup 2019: Robot World Cup XXIII*, ser. Lecture Notes in Computer Science. Sydney: Springer International Publishing, 2019.

[31] A. Wykowska, I. Serena, S. Sam Ge, O. Khatib, J. Perez-Osorio, and S. Anzalone, "International Journal of Social Robotics," available at: https://web.archive.org/web/20230912072607/https://www.springer.com/journal/12369, last viewed: 2023-09-12.

[32] F. Lindner and C. Eschenbach, "Affordances and Affordance Space: A Conceptual Framework for Application in Social Robotics," in *Sociable Robots and the Future of Social Relations*. IOS Press, 2014, pp. 35–45.

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

[34] M. Levihn, "Navigation Among Movable Obstacles in unknown environments," Master's thesis, Georgia Institute of Technology, Atlanta, Georgia, May 2011.

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

[36] G. Wilfong, "Motion planning in the presence of movable obstacles," in *Proceedings of the fourth annual symposium on Computational geometry*, ser. SCG '88. Association for Computing Machinery, 1988, pp. 279–288.

[37] ——, "Motion planning in the presence of movable obstacles," *Annals of Mathematics and Artificial Intelligence*, vol. 3, no. 1, pp. 131–150, 1991.

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

[39] M. Stilman and J. Kuffner, "Navigation among movable obstacles: real-time reasoning in complex environments," in *4th IEEE/RAS International Conference on Humanoid Robots, 2004.*, vol. 1, 2004, pp. 322–341 Vol. 1.

[40] R. Alami, T. Siméon, and J.-P. Laumond, "A geometrical approach to planning manipulation tasks. The case of discrete placements and grasps," in *The fifth international symposium on Robotics research*, H. Miura, Ed. MIT Press, 1990, pp. 453–463.

[41] R. Alami, J.-P. Laumond, and T. Simeon, "Two manipulation planning algorithms," in *WAFR Proceedings of the workshop on Algorithmic foundations of robotics*, K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, Eds. A. K. Peters, Ltd. Natick, MA, USA, 1994, pp. 109–125.

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

[43] M. Stilman, K. Nishiwaki, S. Kagami, and J. J. Kuffner, "Planning and executing navigation among movable obstacles," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 820–826.

[44] ——, "Planning and executing navigation among movable obstacles," *Advanced Robotics*, vol. 21, no. 14, pp. 1617–1634, 2007.

[45] R. H. Wilson, "On Geometric Assembly Planning," Ph.D. dissertation, Stanford University, Mar. 1992.

[46] M. Stilman and J. Kuffner, "Planning among movable obstacles with artificial constraints," *The International Journal of Robotics Research*, vol. 27, no. 11, pp. 1295–1307, 2008.

[47] M. Stilman and J. J. Kuffner, "Planning among movable obstacles with artificial constraints," in *Algorithmic Foundation of Robotics VII: Selected Contributions of the Seventh International Workshop on the Algorithmic Foundations of Robotics*, ser. Springer Tracts in Advanced Robotics, S. Akella, N. M. Amato, W. H. Huang, and B. Mishra, Eds. Springer, 2008, pp. 119–135.

[48] D. Nieuwenhuisen, A. F. v. d. Stappen, and M. H. Overmars, "An effective framework for path planning amidst movable obstacles," Department of Information and Computing Sciences, Utrecht University, Tech. Rep. UU-CS-2006-035, 2006.

[49] D. Nieuwenhuisen, "Path Planning in Changeable Environments," Ph.D. dissertation, Universiteit Utrech, 2007.

[50] J. P. Van den Berg, "Path Planning in Dynamic Environments," Ph.D. dissertation, Universiteit Utrech, 2007.

[51] S. K. Moghaddam and E. Masehian, "Planning Robot Navigation among Movable Obstacles (NAMO) through a Recursive Approach," *Journal of Intelligent & Robotic Systems*, vol. 83, no. 3, pp. 603–634, Sept. 2016.

[52] Z. Meng, H. Sun, K. B. H. Teo, and M. H. Ang, "Active Path Clearing Navigation through Environment Reconfiguration in Presence of Movable Obstacles," in *2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, July 2018, pp. 156–163.

[53] T. G. Dietterich, "An Overview of MAXQ Hierarchical Reinforcement Learning," in *Abstraction, Reformulation, and Approximation*, ser. Lecture Notes in Computer Science, B. Y. Choueiry and T. Walsh, Eds. Berlin, Heidelberg: Springer, 2000, pp. 26–44.

[54] M. Kearns, Y. Mansour, and A. Y. Ng, "A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes," *Machine Learning*, vol. 49, no. 2, pp. 193–208, Nov. 2002.

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

[56] ——, "Integrated task and motion planning in belief space," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, Aug. 2013.

[57] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 265–293, 2021.

[58] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[59] S. M. LaValle and J. J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[60] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, Apr. 2000, pp. 995–1001 vol.2.

[61] R. M. Neal, "MCMC Using Hamiltonian Dynamics," in *Handbook of Markov Chain Monte Carlo*. Chapman and Hall/CRC, 2011.

[62] M. Fiala, "ARTag, a fiducial marker system using digital techniques," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, June 2005, pp. 590–596 vol. 2.

[63] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 3400–3407.

[64] N. Castaman, "A Sampling-Based Tree Planner for Robot Navigation Among Movable Obstacles," Master's thesis, University of Padova, Padova, Italy, July 2016.

[65] N. Cstaman, E. Tosello, and E. Pagello, "A Sampling-Based Tree Planner for Navigation Among Movable Obstacles," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*, June 2016, pp. 1–8.

[66] I. A. Şucan and L. E. Kavraki, "Kinodynamic Motion Planning by Interior-Exterior Cell Exploration," in *Algorithmic Foundation of Robotics VIII: Selected Contributions of the Eight International Workshop on the Algorithmic Foundations of Robotics*, ser. Springer Tracts in Advanced Robotics, G. S. Chirikjian, H. Choset, M. Morales, and T. Murphey, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 449–464.

[67] H. Sun, Z. Meng, and M. H. Ang, "Semantic mapping and semantics-boosted navigation with path creation on a mobile robot," in *2017 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, Nov. 2017, pp. 207–212.

[68] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, June 2011.

[69] B. Renault, J. Saraydaryan, and O. Simonin, "Modeling a Social Placement Cost to Extend Navigation Among Movable Obstacles (NAMO) Algorithms," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, United States, Oct. 2020, pp. 11 345–11 351.

[70] M. Wang, R. Luo, A. O. Onol, and T. Padir, "Affordance-based mobile robot navigation among movable obstacles," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 2734–2740.

[71] J. J. Gibson, *The Ecological Approach to Visual Perception*. New York: Routledge, Oct. 1986.

[72] P. Kaiser, M. Grotz, E. E. Aksoy, M. Do, N. Vahrenkamp, and T. Asfour, "Validation of whole-body loco-manipulation affordances for pushability and liftability," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, Nov. 2015, pp. 920–927.

[73] A. O. Onol, P. Long, and T. Padır, "Contact-Implicit Trajectory Optimization Based on a Variable Smooth Contact Model and Successive Convexification," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 2447–2453.

[74] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026–5033, Oct. 2012.

[75] Y. Mao, D. Dueri, M. Szmuk, and B. Açıkmeşe, "Successive Convexification of Non-Convex Optimal Control Problems with State Constraints," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4063–4069, July 2017.

[76] K. Charalampous, I. Kostavelis, and A. Gasteratos, "Recent trends in social aware robot navigation: A survey," *Robotics and Autonomous Systems*, vol. 93, pp. 85–104, July 2017.

[77] S. F. Chik, C. F. Yeong, E. L. M. Su, T. Y. Lim, Y. Subramaniam, and P. J. H. Chin, "A Review of Social-Aware Navigation Frameworks for Service Robot in Dynamic Human Environments," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 8, no. 11, pp. 41–50–50, Dec. 2016.

[78] J. Cheng, H. Cheng, M. Q.-H. Meng, and H. Zhang, "Autonomous Navigation by Mobile Robots in Human Environments: A Survey," in *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Dec. 2018, pp. 1981–1986.

[79] R. Möller, A. Furnari, S. Battiato, A. Härmä, and G. M. Farinella, "A survey on human-aware robot navigation," *Robotics and Autonomous Systems*, vol. 145, p. 103837, Nov. 2021.

[80] C. Mavrogiannis, F. Baldini, A. Wang, D. Zhao, P. Trautman, A. Steinfeld, and J. Oh, "Core Challenges of Social Robot Navigation: A Survey," *ACM Transactions on Human-Robot Interaction*, Feb. 2023.

[81] E. T. Hall, *The hidden dimension : man's use of space in public and private*. London : Bodley Head, 1966.

[82] F. Lindner and C. Eschenbach, "Towards a Formalization of Social Spaces for Socially Aware Robots," in *Spatial Information Theory*, ser. Lecture Notes in Computer Science, M. Egenhofer, N. Giudice, R. Moratz, and M. Worboys, Eds. Springer Berlin Heidelberg, 2011, pp. 283–303.

[83] F. Ostermann and S. Timpf, "Modelling space appropriation in public parks," in *Ostermann, F; Timpf, S (2007). Modelling space appropriation in public parks. In: Wachowicz,*

M. Proceedings of 10th AGILE International Conference on Geographic Information Science. Aalborg, DK: Aalborg University, 1-7., M. Wachowicz, Ed. Aalborg, DK: Aalborg University, 2007, pp. 1–7.

[84] B. Okal and T. Linder, "Pedestrian Simulator," available at: https://github.com/srl-freiburg/pedsim_ros, last viewed: 2024-01-25.

[85] E. A. Sisbot, L. F. Marin, and R. Alami, "Spatial reasoning for human robot interaction," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2007, pp. 2281–2287.

[86] K. L. Koay, E. A. Sisbot, D. S. Syrdal, M. L. Walters, K. Dautenhahn, and R. Alami, "Exploratory Study of a Robot Approaching a Person in the Context of Handing Over an Object," in *AAAI Spring Symposium: Multidisciplinary Collaboration for Socially Assistive Robotics*, 2007.

[87] E. A. Sisbot, L. F. Marin-Urias, X. Broquère, D. Sidobre, and R. Alami, "Synthesizing Robot Motions Adapted to Human Presence," *International Journal of Social Robotics*, vol. 2, no. 3, pp. 329–343, Sept. 2010.

[88] J. Mainprice, E. A. Sisbot, T. Simeon, and R. Alami, "Planning Safe and Legible Hand-over Motions for Human-Robot Interaction," in *Proceedings of the IARP/IEEE-RAS/EURON workshop on technical challenges for dependable robots in human environments*, 2010.

[89] J. Mainprice, E. Akin Sisbot, L. Jaillet, J. Cortés, R. Alami, and T. Siméon, "Planning human-aware motions using a sampling-based costmap planner," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 5012–5017.

[90] E. A. Sisbot and R. Alami, "A Human-Aware Manipulation Planner," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1045 – 1057, Oct. 2012.

[91] J. Mainprice, M. Gharbi, T. Siméon, and R. Alami, "Sharing effort in planning human-robot handover tasks," in *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, Sept. 2012, pp. 764–770.

[92] M. Gharbi, S. Lemaignan, J. Mainprice, and R. Alami, "Natural interaction for object hand-over," in *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2013, pp. 401–401.

[93] J. Waldhart, M. Gharbi, and R. Alami, "Planning handovers involving humans and robots in constrained environment," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept. 2015, pp. 6473–6478.

[94] J. Mainprice and D. Berenson, "Human-Robot Collaborative Manipulation Planning Using Early Prediction of Human Motion," in *IEEE International Conference on Intelligent Robots and Systems*, Nov. 2013.

[95] P. A. Lasota and J. A. Shah, "Analyzing the Effects of Human-Aware Motion Planning on Close-Proximity Human–Robot Collaboration," *Human Factors*, vol. 57, no. 1, pp. 21–33, Feb. 2015.

[96] S. Pellegrinelli, F. L. Moro, N. Pedrocchi, L. Molinari Tosatti, and T. Tolio, "A probabilistic approach to workspace sharing for human–robot cooperation in assembly tasks," *CIRP Annals*, vol. 65, no. 1, pp. 57–60, Jan. 2016.

[97] S. Pellegrinelli, A. Orlandini, N. Pedrocchi, A. Umbrico, and T. Tolio, "Motion planning and scheduling for human and industrial-robot collaboration," *CIRP Annals*, vol. 66, no. 1, pp. 1–4, Jan. 2017.

[98] V. V. Unhelkar, P. A. Lasota, Q. Tyroller, R.-D. Buhai, L. Marceau, B. Deml, and J. A. Shah, "Human-Aware Robotic Assistant for Collaborative Assembly: Integrating Human Motion Prediction With Planning in Time," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2394–2401, July 2018.

[99] R. Alami, M. Gharbi, B. Vadant, R. Lallement, and A. Suarez, "On human-aware task and motion planning abilities for a teammate robot," in *Human-Robot Collaboration for Industrial Manufacturing Workshop, RSS 2014*, UC Berkeley, United States, July 2014.

[100] M. Faroni, M. Beschi, S. Ghidini, N. Pedrocchi, A. Umbrico, A. Orlandini, and A. Cesta, "A Layered Control Approach to Human-Aware Task and Motion Planning for Human-Robot Collaboration," in *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. Naples, Italy: IEEE, Aug. 2020, pp. 1204–1210.

[101] A. Farinelli, L. Iocchi, and D. Nardi, "Multirobot systems: a classification focused on coordination," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 34, no. 5, pp. 2015–2028, Oct. 2004.

[102] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach, Global Edition*, 3rd ed. Pearson, 2016.

[103] L. Iocchi, D. Nardi, and M. Salerno, "Reactivity and Deliberation: A Survey on Multi-Robot Systems," in *Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, ser. Lecture Notes in Computer Science, M. Hannebauer, J. Wendler, and E. Pagello, Eds. Berlin, Heidelberg: Springer, 2001, pp. 9–32.

[104] Y. U. Cao, A. S. Fukunaga, and A. Kahng, "Cooperative Mobile Robotics: Antecedents and Directions," *Autonomous Robots*, vol. 4, no. 1, pp. 7–27, Mar. 1997.

[105] Z. Yan, N. Jouandeau, and A. A. Cherif, "A Survey and Analysis of Multi-Robot Coordination," *International Journal of Advanced Robotic Systems*, vol. 10, no. 12, p. 399, Dec. 2013.

[106] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. Kumar, E. Boyarski, and R. Barták, "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks," *SOCS*, 2019.

[107] S. Koenig, "mapf.info," available at: http://mapf.info/, last viewed: 2024-01-25.

[108] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, Feb. 2015.

[109] Z. Feng, G. Hu, Y. Sun, and J. Soon, "An overview of collaborative robotic manipulation in multi-robot systems," *Annual Reviews in Control*, vol. 49, pp. 113–127, Jan. 2020.

[110] L. Lindzey, R. A. Knepper, H. Choset, and S. S. Srinivasa, "The Feasible Transition Graph: Encoding Topology and Manipulation Constraints for Multirobot Push-Planning," in *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, ser. Springer Tracts in Advanced Robotics, H. L. Akin, N. M. Amato, V. Isler, and A. F. van der Stappen, Eds. Cham: Springer International Publishing, 2015, pp. 301–318.

[111] M. Levihn, T. Igarashi, and M. Stilman, "Multi-robot multi-object rearrangement in assignment space," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, pp. 5255–5261.

[112] N. Oyama, Z. Liu, L. B. Gueta, and J. Ota, "Rearrangement task of multiple robots using task assignment applicable to different environments," in *2010 IEEE International Conference on Robotics and Biomimetics*, Dec. 2010, pp. 300–305.

[113] ——, "Task apportionment in a rearrangement problem of multiple mobile robots," *Advanced Robotics*, vol. 27, no. 2, pp. 93–107, Feb. 2013.

[114] R. Inoue, N. Fujii, R. Takano, and J. Ota, "Rearrangement of multiple objects by a robot group having a multi-task function," in *2008 IEEE International Conference on Robotics and Biomimetics*, Feb. 2009, pp. 2013–2018.

[115] R. Shome and K. E. Bekris, "Synchronized multi-arm rearrangement guided by mode graphs with capacity constraints," in *Algorithmic Foundations of Robotics XIV*, ser. Springer Proceedings in Advanced Robotics, S. M. LaValle, M. Lin, T. Ojala, D. Shell, and J. Yu, Eds. Springer International Publishing, 2021, pp. 243–260.

[116] V. N. Hartmann, A. Orthey, D. Driess, O. S. Oguz, and M. Toussaint, "Long-horizon multi-robot rearrangement planning for construction assembly," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 239–252, 2023.

[117] J. Motes, T. Chen, T. Bretl, M. M. Aguirre, and N. M. Amato, "Hypergraph-based multi-robot task and motion planning," *IEEE Transactions on Robotics*, 2023.

[118] L. Antonyshyn, J. Silveira, S. Givigi, and J. Marshall, "Multiple Mobile Robot Task and Motion Planning: A Survey," *ACM Computing Surveys*, vol. 55, no. 10, pp. 213:1–213:35, Feb. 2023.

[119] J. Motes, R. Sandström, H. Lee, S. Thomas, and N. M. Amato, "Multi-Robot Task and Motion Planning With Subtask Dependencies," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3338–3345, Apr. 2020.

[120] A. Thomas, F. Mastrogiovanni, and M. Baglietto, "Towards Multi-Robot Task-Motion Planning for Navigation in Belief Space," *ECAI-STAIRS 2020*, Oct. 2020.

[121] W. N. Tang, S. D. Han, and J. Yu, "Computing high-quality clutter removal solutions for multiple robots," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 7963–7970.

[122] J.-C. Latombe, *Robot Motion Planning*, ser. The Springer International Series in Engineering and Computer Science. Springer US, 1991.

[123] M. T. Mason, *Mechanics of Robotic Manipulation*. Cambridge, Mass: A Bradford Book, Aug. 2001.

[124] K. M. Lynch and M. T. Mason, "Stable Pushing: Mechanics, Controllability, and Planning," *The International Journal of Robotics Research*, vol. 15, no. 6, pp. 533–556, Dec. 1996.

[125] M. Erdmann, "An Exploration of Nonprehensile Two-Palm Manipulation," *The International Journal of Robotics Research*, vol. 17, no. 5, pp. 485–503, May 1998.

[126] G. Morris and L. Haynes, "Robotic assembly by constraints," in *1987 IEEE International Conference on Robotics and Automation Proceedings*, vol. 4, Mar. 1987, pp. 1507–1515.

[127] J. Morrow and P. Khosla, "Manipulation task primitives for composing robot skills," in *Proceedings of International Conference on Robotics and Automation*, vol. 4, Apr. 1997, pp. 3354–3359 vol.4.

[128] T. Igarashi and M. Stilman, "Homotopic Path Planning on Manifolds for Cabled Mobile Robots," in *Algorithmic Foundations of Robotics IX*, ser. Springer Tracts in Advanced Robotics, B. Siciliano, O. Khatib, F. Groen, D. Hsu, V. Isler, J.-C. Latombe, and M. C. Lin, Eds., vol. 68. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–18.

[129] P. Jansen, R. Goud, and M. Wener, "TIOBE Index," available at https://www.tiobe.com/tiobe-index/, last viewed: 2024-01-25.

[130] P. Carbonnelle, "PYPL PopularitY of Programming Language index," available at https://pypl.github.io/PYPL.html, last viewed: 2024-01-25.

[131] P. S. Foundation, "Sunsetting Python 2," available at https://www.python.org/doc/sunset-python-2/, last viewed: 2024-01-25.

[132] A. Foisy and V. Hayward, "A safe swept volume method for collision detection," *International Journal of Robotic Research - IJRR*, Jan. 1994.

[133] K. A. Hart and J. J. Rimoli, "Generation of statistically representative microstructures with direct grain geometry control," *Computer Methods in Applied Mechanics and Engineering*, vol. 370, 2020.

[134] J. Ferber, *LES SYSTEMES MULTI-AGENTS. Vers une intelligence collective*. Paris: Dunod, 1995.

[135] F. Lindner, "A conceptual model of personal space for human-aware robot activity placement," *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5770–5775, Sept. 2015.

[136] Y. Jiang, M. Lim, and A. Saxena, "Learning object arrangements in 3d scenes using human context," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ser. ICML'12. Omnipress, 2012, pp. 907–914.

[137] Y. Jiang and A. Saxena, "Hallucinating Humans for Learning Robotic Placement of Objects," in *Experimental Robotics: The 13th International Symposium on Experimental Robotics*, ser. Springer Tracts in Advanced Robotics, J. P. Desai, G. Dudek, O. Khatib, and V. Kumar, Eds. Heidelberg: Springer International Publishing, 2013, pp. 921–937.

[138] Y. Jiang, H. Koppula, and A. Saxena, "Hallucinated humans as the hidden context for labeling 3d scenes," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2993–3000.

[139] Y. Jiang and A. Saxena, "Infinite latent conditional random fields for modeling environments through humans," in *Robotics: Science and Systems IX*. Robotics: Science and Systems Foundation, 2013.

[140] S.-H. Zhang, S.-K. Zhang, Y. Liang, and P. Hall, "A Survey of 3D Indoor Scene Synthesis," *Journal of Computer Science and Technology*, vol. 34, no. 3, pp. 594–608, May 2019.

[141] L.-F. Yu, S. K. Yeung, C.-K. Tang, D. Terzopoulos, T. F. Chan, and S. Osher, "Make it home: automatic optimization of furniture arrangement," *ACM Trans. Graph.*, vol. 30, p. 86, 2011.

[142] M. Mitton and C. Nystuen, *Residential Interior Design: A Guide to Planning Spaces*, 3rd ed. New York, NY: Wiley, Apr. 2016.

[143] F. D. Ching and C. Binggeli, *Interior Design Illustrated*, 4th ed. New York, NY: Wiley, Feb. 2018.

[144] S. Qi, Y. Zhu, S. Huang, C. Jiang, and S.-C. Zhu, "Human-centric indoor scene synthesis using stochastic grammar," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5899–5908.

[145] D. V. Lu, D. Hershberger, and W. D. Smart, "Layered costmaps for context-sensitive navigation," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept. 2014, pp. 709–715.

[146] M. Mielle, M. Magnusson, H. Andreasson, and A. J. Lilienthal, "SLAM auto-complete: Completing a robot map using an emergency map," in *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, Oct. 2017, pp. 35–40.

[147] French Interior Ministry, "Arrete du 25 juin 1980 portant approbation des dispositions generales du reglement de securite contre les risques d'incendie et de panique dans les etablissements recevant du public (ERP)." Sept. 2019, livre II, Section 9, Sous-section 1, Article CO 36. Available at https://www.legifrance.gouv.fr/loda/id/LEGITEXT000020303557, last viewed: 2024-01-25.

[148] P. K. Saha, G. Borgefors, and G. S. d. Baja, *Skeletonization: Theory, Methods and Applications*. Academic Press, June 2017.

[149] R. Fabbri, L. D. F. Costa, J. C. Torelli, and O. M. Bruno, "2D Euclidean Distance Transform Algorithms: A Comparative Survey," *ACM Comput. Surv.*, vol. 40, no. 1, pp. 2:1–2:44, Feb. 2008.

[150] T. Y. Zhang and C. Y. Suen, "A Fast Parallel Algorithm for Thinning Digital Patterns," *Commun. ACM*, vol. 27, no. 3, pp. 236–239, Mar. 1984.

[151] Z. Guo and R. W. Hall, "Parallel thinning with two-subiteration algorithms," *Communications of the ACM*, vol. 32, no. 3, pp. 359–373, Mar. 1989.

[152] H. Blum, "A Transformation for Extracting New Descriptors of Shape," in *Models for the perception of speech and visual form*, W. Wathen-Dunn, Ed. Cambridge: MIT Press, 1967, pp. 362–380.

[153] M. J.F.M. Johan, "DINED - anthropometric database," *4TU.Centre for Research Data*, 2018.

[154] S. M. LaValle, "Planning Algorithms," May 2006.

[155] D. Harabor and A. Grastien, "Online graph pruning for pathfinding on grid maps," in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, ser. AAAI'11. AAAI Press, 2011, pp. 1114–1119.

[156] J. Ferber, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*, 1st ed. Addison-Wesley Longman Publishing Co., Inc., 1999.

[157] K. Ellis, H. Zhang, D. Stoyanov, and D. Kanoulas, "Navigation Among Movable Obstacles with Object Localization using Photorealistic Simulation," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2022, pp. 1711–1716.

[158] Z. Kai, E. Lucet, J. Alexandre dit Sandretto, and D. Filliat, "Navigation among movable obstacles using machine learning based total time cost optimization," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

FOLIO ADMINISTRATIF

THESE DE L'INSA LYON, MEMBRE DE L'UNIVERSITE DE LYON

NOM : RENAULT DATE de SOUTENANCE : 19/12/2023

Prénoms : Benoit Louis Robert

TITRE : NAvigation en milieu MOdifiable (NAMO) étendue à des contraintes sociales et multi-robots

NATURE : Doctorat Numéro d'ordre : 2023ISAL0105

Ecole doctorale : INFORMATIQUE ET MATHEMATIQUES DE LYON (InfoMaths)

Spécialité : Informatique

RESUME :

Alors que les robots deviennent toujours plus présents dans les environnements humains, endossant toujours plus de tâches telles que le nettoyage, la surveillance ou encore le service en salle, leurs limites actuelles n'en deviennent que plus évidentes. Une de ces limites concerne leur capacité à naviguer en présence d'obstacles: ils chercheront systématiquement à les éviter, et resteront bloqués à défaut.
Ce constat a mené à la création d'algorithmes de NAvigation en milieu MOdifiable (NAMO), devant permettre aux robots de manipuler les obstacles pour faciliter leurs déplacements. Néanmoins, ces algorithmes ont été conçus sous l'hypothèse qu'un seul robot agîsse dans l'environnement, biaisant les algorithmes à n'optimiser que son seul coût de déplacement – sans considération pour les humains ou d'autres robots. S'il est souhaitable que les robots puissent bénéficier de la capacité humaine à déplacer des obstacles, ils doivent néamoins le faire dans le respect des normes et règles sociales humaines.
Nous avons donc étendu le problème de NAMO pour prendre en compte ces nouveaux aspects sociaux et multi-robots. En nous basant sur le concept d'espaces d'affordance, nous avons développé un modèle de coût d'occupation sociale permettant d'évaluer l'impact des objets déplacés sur la navigabilité de l'environnement. Nous avons implémenté (et amélioré) des algorithmes NAMO de référence, dans notre outil de simulation open source, puis les avons modifiés afin qu'ils puissent trouver un compromis entre coût de déplacement et coût d'occupation des obstacles manipulés – résultant en une amélioration de la navigabilité. Nous avons également développé une stratégie de coordination permettant d'exécuter ces mêmes algorithmes tels quels, sur plusieurs robots en parallèle, en absence de communication explicite, tout en préservant la garantie d'absence de collisions; vérifiant la pertinence de notre modèle de coût social en présence effective d'autres robots. Ces travaux constituent les premiers pas d'une NAMO Sociale et Multi-Robots.

MOTS-CLÉS : NAMO, Navigation Among Movable Obstacles, Multi-Robot Systems, Multi-Agent Systems, Social Robotics, Path Planning, Task and Motion Planning

Laboratoire (s) de recherche : CITI (INRIA – INSA Lyon)

Directeur de thèse: SIMONIN Olivier, Jacques SARAYDARYAN (co-encadrant)

Président de jury :

Dugdale, Julie	Professeur des Universités	Université de Grenoble, LIG	

Composition du jury :

Mathieu, Philippe	Professeur des Universités	Université de Lille, CRISTAL	Rapporteur
Michel, Fabien	Maître de Conférences HDR	Université Montpellier 2, LIRMM	Rapporteur
Alami, Rachid	Directeur de Recherche émérite	CNRS Toulouse, LAAS	Examinateur
Simonin, Olivier	Professeur des Universités	INSA-Lyon, CITI	Directeur de thèse
Saraydaryan, Jacques	Enseignant Chercheur	CPE Lyon, CITI	Co-encadrant