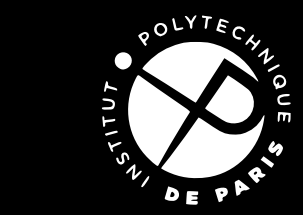
Online Distributed Motion Planning for Mobile Multi-robot Systems

José Mendes Filho



INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2019IPPAE007

Thèse de doctorat

Online Distributed Motion Planning for Mobile Multi-robot Systems

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée au CEA Saclay

Ecole doctorale n° 626 de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Signal, Images, Automatique et Robotique

Thèse présentée et soutenue à Palaiseau, le 19/12/2019, par

M. JOSÉ MENDES FILHO

Composition du Jury :

Roland LENAIN Directeur de Recherche, IRSTEA	Président du Jury
Abdel-Illah MOUADDIB Professeur, Université de Caen	Rapporteur
Anne SPALANZANI Maitre de Conférence HDR, Université de Grenoble	Rapporteur
Alain MICAELLI Directeur de Recherche, CEA Saclay	Examinateur
David FILLIAT Professeur, ENSTA Paris (U2IS)	Directeur de thèse
Eric LUCET Ingénieur de Recherche, CEA Saclay	Co-encadrant

I would like to dedicate this thesis to the few but very special Masters[1] life had favoured me with, whether in the form of people, things or events.

[1] Masters as in "Master-Disciple Relationship"

Acknowledgements

First, I would like to acknowledge the immense help my advisers, David Filliat and Eric Lucet, gave me during these years of work. I especially appreciate their patience during the fourth year of my thesis, where my work was considerably sparse and irregular.

I would like to thank as well both laboratories at ENSTA Paris and CEA Saclay and their whole staff for making this work possible. Among all my colleges, I owe a special thanks to Selma Kchir and Djibril Diallo, for making working at CEA a lot more fun then it would have been without their presence. I am very thankful for the life long friendship we formed at office 33.

I could not forget to thank also my flatmate Titouan Boulmier for being always there to hear me complain or talk in excitement about this thesis.

Last but not least, I thank my incredible, eternal friend and ex-flatmate Graziela Cupertino for so much support along the way and for keep pushing me to finish this dissertation.

Abstract

This thesis aims to study and develop an approach for solving the motion planning problem of a group of wheeled mobile robots in a realistic environment. Mainly we propose a distributed mathematical programming approach associated with a receding horizon method for the open-loop trajectory generation as well as a modified model predictive control (MPC) for the closed-loop stabilization. In this approach, perception, trajectory planning, and execution are interleaved and can be performed onboard each robot independently, as they evolve through their workspace. It ensures respect of several types of constraints, namely obstacle avoidance, bounded velocities and accelerations, nonholonomic constraints, and inter-robot collision avoidance. The robots belonging to the multi-robot system exchange information on their intended trajectories and converge individually to optimal non-conflicting trajectories.

Furthermore, some work towards integrated task and motion planning by a hierarchical method is presented. The objective was to achieve a complete framework for robust, highly autonomous mobile robot motion.

Experiments both in simulation and with real nonholonomic unicycle-like vehicles were conducted. They allowed us to analyze the impact of parameters on key figures such as computation time, obstacle avoidance, inter-robot collision avoidance, and travel time. Results also show the quality of robot motion in situations where dynamics, the uncertainties about robot localization, and communication delays are real and meaningful.

Overall, this study indicates that the proposed approach could be used in real systems where uncertainty about the world state, communication delays, limited onboard computation power, strong dynamics, and other usually challenging to overcome phenomena are all present.

Résumé

Ce travail de doctorat a été financé et réalisé au Laboratoire de Robotique Interactive de l'Institut List du CEA, en partenariat avec l'Unité d'Informatique et d'Ingénierie des Systèmes de l'ENSTA Paris. Cette recherche s'inscrit dans le cadre des travaux du CEA sur la navigation précise de véhicules autonomes dans des environnements où coexistent des êtres humains et des robots mobiles tout en bénéficiant de l'expertise de l'ENSTA Paris en matière de navigation, de perception et de modélisation sémantique de l'environnement.

Contexte

La pertinence d'une telle recherche peut être mise en valeur par l'analyse de trois contextes différents et leurs particularités à la fin des années 2010. D'un point de vue **social**, la présence des robots et de l'IA (Intelligence Artificielle) dans l'espace de travail prend une ampleur significative et les prédictions sur l'impact social de l'automatisation sont diverses, mais alertent souvent sur la vulnérabilité des travailleurs à des postes qui exigent peu de qualifications. Logiquement, avec une augmentation des tâches réalisables par des systèmes artificiels autonomes, la vulnérabilité de ces travailleurs ne fera qu'augmenter. Par rapport à cela, l'étude de comment générer des mouvements afin d'avoir des robots plus collaboratifs (plus faciles à déployer et à utiliser) peut diminuer la demande de formation pour ces travailleurs et par conséquent une atténuation de leur vulnérabilité.

D'un point de vue **économique**, dans le domaine particulier de la logistique, l'utilisation de robots a connu une croissance rapide au cours des dernières années motivée par 1) la nécessité de réduire les coûts de la chaîne d'approvisionnement et 2) l'augmentation réglementaire de la sécurité et du confort des travailleurs. Cette thèse sur la navigation mobile multi-robots peut avoir une utilité directe dans l'amélioration des opérations liées aux entrepôts telles que la préparation des commandes, l'optimisation du stockage et de la distribution des produits, tout en améliorant les paramètres de sécurité pour les travailleurs.

Finalement, d'un point de vue **scientifique**, une plate-forme robotique mobile est perçue comme un système dont la capacité à changer d'état, sa capacité à soutenir les efforts et son énergie disponible sont limitées. Des contraintes sont donc nécessaires pour générer

des trajectoires, limitant certaines transitions entre deux configurations. Ce problème est désigné par différents termes, les plus courants étant la planification de mouvement, la planification de trajectoire et la planification kinodynamique. Divers critères pour la classification des approches de planification de trajectoire existent. Les plus courants sont les suivants :

- La qualité de la solution ou son optimalité ;
- Le coût de calcul ;
- La complétude - la garantie de trouver une solution si au moins une existe ;
- La propriété d'exécution *en ligne* – la planification et l'exécution sont simultanées ;
- La propriété de *temps réel* – l'algorithme de planification peut s'exécuter à une fréquence suffisamment élevée et consistante pour que l'hypothèse d'un environnement statique pendant une itération soit raisonnable.

Dans le cas de la planification pour un système composé de plusieurs robots (MRS), on parle aussi d'approches *centralisées* et *décentralisées*. La quête d'approches qui permettent d'équilibrer de façon satisfaisante plusieurs de ces propriétés en même temps reste un sujet de recherche ouvert et de plus en plus étudié.

Étant donnés ces trois contextes, dans ce travail de thèse, nous nous concentrons sur l'étude et le développement d'une approche pour résoudre le problème de planification de mouvements d'un groupe de robots mobiles à roues différentielles en environnement opérationnel réaliste.

Planification de trajectoire multi-robots

Nous proposons principalement une approche basée sur l'optimisation distribuée associée à une méthode de type fenêtre glissante pour la génération de trajectoire en boucle ouverte, ainsi qu'une loi de commande prédictive (MPC) pour la stabilisation du système en boucle fermée. Dans cette approche, la perception, la planification de trajectoire et son exécution sont combinées et peuvent être réalisées indépendamment par le contrôleur de chacun des robots, au fur et à mesure qu'ils évoluent dans leur espace de travail.

Plus spécifiquement, nous modélisons la génération de trajectoire comme un problème d'optimisation non-linéaire sous contraintes. Ce problème est définit pour un horizon de temps fixé appelé T_p (période de planification). Cette durée T_p "glisse" selon l'évolution

du robot d'une valeur fixée appelé T_c (période de calcul). T_c est l'intervalle pour lequel une solution au problème d'optimisation doit être trouvée.

Les contraintes d'optimisation peuvent être des équations ou des inéquations. Les fonctions d'objectifs sont choisies afin d'attirer les robots vers leurs configurations désirées – la minimisation du temps de trajet ou de la distance restante vers l'objectif. L'approche garantit le respect de plusieurs types de contraintes, à savoir l'évitement des obstacles, la limitation des vitesses et des accélérations, les contraintes non-holonomes et l'évitement des collisions inter-robots.

Cette optimisation se déroule en deux étapes différentes :

- Une première étape qui ignore les contraintes liées aux autres robots, appelées les contraintes de couplage. Les contraintes liées au modèle cinématique de chaque véhicule, leur limitation de vitesse et d'accélération, et l'évitement d'obstacles sont néanmoins pris en compte. Cela permet aux robots de trouver une première trajectoire qui sera alors partagée avec d'autres robots à proximité;

- La deuxième étape intègrera, en tant que contraintes de couplage, les trajectoires des autres robot calculées dans la première étape. La trajectoire issue de l'étape numéro deux sera celle utilisée comme trajectoire de référence par la boucle de stabilisation du système robotique.

Ces deux étapes combinées doivent être effectuées en un temps inférieur à T_c. Et T_c doit être suffisamment petit pour que les trajectoires des différents robots convergent vers des solutions sans collisions d'un point de vue global.

Ainsi, les robots appartenant au système multi-robots échangent des informations sur leurs trajectoires envisagées et convergent individuellement vers des trajectoires optimales sans conflit.

Cette approche à fenêtre glissante est appropriée tant que les robots se retrouvent loin du voisinage de leur configuration but. Au moment de l'arrivée à l'objectif, une formulation particulière du problème d'optimisation est nécessaire afin de respecter les contraintes sur la configuration finale du robot. Nous regroupons ces idées et la façon de les implémenter sous l'acronyme DRHMP (*Distributed Receding Horizon Motion Planning* ou Planification de Mouvements Distribuée sur un Horizon Glissant).

L'approche DRHMP présente des avantages par rapport à des méthodes plus classiques grâce à sa flexibilité d'intégrer de nouvelles contraintes, son calcul en ligne et en temps réel et son indépendence aux trajectoires pré-definies. La complétude de l'approche n'est pas garantie, mais des actions permenttant de remédier à un échec de planification sont discutées dans les conclusions de ce travail.

Commande prédictive

Une fois une trajectoire de référence générée, une commande de type predictive, continue, non-linéaire et généralisée (NCGPC) a été utilisée pour la stabilisation du système. Pour exploiter l'intégralité de la trajectoire de référence, nous avons proposé une modification de cette loi de commande. La nouvelle commande (appelée NCGPC-M pour NCGPC modifiée) a été créée en remplaçant l'extrapolation de la sortie de référence utilisée dans l'approche de base NCGPC par la prédiction de notre planificateur de mouvement DRHMP.

Nous avons comparé ces deux approches entre elles ainsi qu'avec une méthode basée sur la transformation du modèle cinématique du robot en système chaîné et montré la supériorité de notre approche en terme de précision de suivi de trajectoire dans les cas fortement dynamiques.

Planification de tâches

Finalement, des travaux en vue d'une planification intégrée des tâches[2] et des mouvements par une méthode hiérarchique sont présentés. L'objectif est d'aboutir à une méthode complète de planification de mouvements robuste et hautement autonome des robots mobiles.

En effet, un système robotique qui vise à être utilisé dans des cas d'usage réels doit tenir compte de deux sources d'incertitudes lors de la planification des tâches : la première concernant l'estimation de l'état de l'environnement (incertitude liée à la perception) et la seconde concernant les résultats des actions (incertitude liée à la prédictibilité). D'un point de vue de la théorie de la décision, ce problème est simplement décrit comme de la *planification sous incertitude*. Le problème pour lequel nous avons développé les approches DRHMP et NCGPC-M présente une grande quantité de ces deux sources d'incertitude, comme la plupart des scénarios réels.

Une façon possible de modéliser les problèmes de planification sous incertitude est d'utiliser des POMDPs (*Partially Observable Markov Decision Processes*). Les POMDPs modélisent un processus de décision pour lequel on suppose que les résultats des actions sont en partie aléatoires et en partie sous le contrôle d'un décideur (un acteur), mais que l'acteur ne peut pas observer directement l'état de l'environnement dans lequel il évolue. Il doit plutôt maintenir une distribution de probabilités sur l'ensemble des états possibles, en fonction d'un ensemble d'observations et de probabilités de ces observations, et du processus de décision de Markov (PDM) sous-jacent. La résolution des POMDP dans des

[2] une séquence d'actions ou plan d'action

scénarios réels est habituellement difficile, et c'est pourquoi de nombreuses approches ont été proposées pour réduire la complexité de ces problèmes.

Dans ce travail de thèse, nous nous basons sur une approche particulière appelée HPN pour *Hierarchical Planning in the Now*. Dans cette approche, on évite de chercher des solutions optimales au POMDP en construisant une approximation déterministe de la dynamique de l'environnement, en cherchant un plan d'action séquentiel sans ramification, en exécutant le plan tout en observant l'état de l'environnement pour déceler les déviations des résultats attendus et en planifiant de nouveau lorsque des écarts se produisent. En outre, pour faire face à l'incertitude concernant l'état actuel, la planification se fait dans l'espace de croyance.

Résultats expérimentaux

Afin d'évaluer l'ensemble de ces développements, des expériences en simulation et avec des véhicules réels de type monocycle non-holonomes ont été menées. Elles ont permis d'analyser l'impact des paramètres sur des critères déterminants tels que le temps de calcul, l'évitement des obstacles, l'évitement des collisions inter-robots et le temps de déplacement. Ces résultats démontrent également la qualité du mouvement du robot dans des situations où la dynamique, les incertitudes sur la localisation du robot et les délais de communication sont réels et significatifs.

Finalement, cette étude montre que l'approche proposée pourrait être utilisée sur des systèmes réels où l'incertitude sur l'état de l'environnement, les retards de communication, la puissance de calcul embarquée limitée, la forte dynamique et d'autres phénomènes habituellement difficiles à surmonter sont tous présents.

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

$\mathscr{C}$	communicated information among robots
C	differentiability/smoothness class
$\mathscr{D}$	set of robots inside the conflict zone of a given robot
$\mathscr{E}$	set of entities described in the belief state
e	error variable
$\mathscr{F}$	Frénet frame
f	function describing the motion of the system
$\hbar$	numeric differentiation step
h	inequation constraints
k	non-negative integer variable
L	Lie derivative
m	non-negative integer variable
$\mathscr{N}$	normal distribution
N	natural number (number of samples)
n	non-negative integer variable
p	non-negative integer variable, or 2D point
q	configuration of the system

T time interval

t continuous time variable

u control input of the system

v linear velocity

x state of the system

z flat output

Greek Symbols

α scaling/multiplication factor

ϵ tolerance value

Γ set of trajectory parameters

γ goal at the task planning level

ω angular velocity

π $\simeq 3.14\ldots$

Φ cumulative distribution function

φ inverse flat output function

ρ relative degrees of a nonlinear MIMO system

Σ summation

σ standard deviation

τ alternative time variable, or logical test of a fluent

θ dynamic system parameters

Θ exteroceptive observation

Superscripts

$\dot{\square}$ first order time derivative

$\ddot{\square}$ second order time derivative

$\square^{(*)}$ *-th derivative

$\square^{T}$ transpose of a matrix

Subscripts

$\square_0$ initial (zero)

$\square_f$ final

$\square_{i,j}$ vector/matrix indexes

$\square_{\max}$ maximum value

$\square_{\min}$ minimum value

$\square_{\text{ref}}$ reference value

Other Symbols

$\wedge$ logical *and*

$\int$ integral sign

$\neg$ logical *not*

∂ partial derivative

$\propto$ proportional to

Acronyms / Abbreviations

3Ds Dull, Dirty and Dangerous

4Ds Dull, Dirty, Dangerous and *Difficult*

C-space Configuration Space

C_{free} Free Subset of the Configuration Space

C_{obst} Obstructed Subset of the Configuration Space

CDF Cumulative Distribution Function

CEA Alternative Energies and Atomic Energy Commission

CFSQP Constraint Feasible Sequential Quadratic Programming

CLDSSPP α-cost-likelihood DSSPP

COBYLA Constrained Optimization by Linear Approximation

DDS Data Distribution Service

DecPOMDP Decentralized Partially Observable Markov Decision Process

DOF degrees of freedom

DRHMP Distributed Receding Horizon Motion Planning

DSSPP Determined Stochastic Shortest-Path Problem

HPN Hierarchical Planning in the Now

HTN Hierarchical Task Network

IPOPT Interior Point Optimizer

IRD Inter-Robot Distance

ISRES Improved Stochastic Ranking Evolution Strategy

MCT Maximum Computation Time

MDP Markov Decision Process

MIMO Multiple Input Multiple Output

MMP Multi-robot Motion Planning

MPC Model Predictive Control

MP Motion Planning

MRS Multi-robot System

NAMO Navigation Among Movable Obstacles

NCGPC-M Modified Nonlinear Continuous Generalized Predictive Control

NCGPC Nonlinear Continuous Generalized Predictive Control

NLopt Nonlinear Optimization Software for C/C++

NLP Nonlinear Optimization Problem

OECD Organization for Economic Co-operation and Development

PDDL Planning Domain Definition Language

PNM Probability Near Mode

PNVN Probability Near Value for Normal Distribution

PNVVM Probability Near Value for Von Mises Distribution

POMDP Partially Observable Markov Decision Process

PSWD Precisely Somewhere Within Delta

RCP Regression Cost Problem

RMS Root Mean Square

ROS Robot Operating System

RPC Region Probably Clear

SLSQP Sequential Least Squares Quadratic Programming

SSPP Stochastic Shortest-Path Problem

STRIPS Stanford Research Institute Problem Solver

TEB Timed-Elastic-Bands

TWD There Within Delta

WMS Warehouse Management System

XDE CEA XDE physics simulator

Chapter 1

Introduction

This doctoral work was funded and carried out at the Interactive Robotics Laboratory of the CEA List Institute, conducted in partnership with the Computer Science and Systems Engineering Unit of ENSTA Paris. This research is in line with the CEA's work on the precise navigation of autonomous vehicles in environments where humans and mobile robots coexist while benefiting from the expertise of ENSTA Paris in terms of navigation, perception and semantic modeling of the environment. The aim is to automate mobile platforms for the efficient supply of assembly lines while guaranteeing the safety of people and property. It concerns the autonomous and highly dynamic evolution of a fleet of mobile robots in the presence of static obstacles (walls, shelving, stored crates, etc.) and dynamic obstacles such as workers and autonomous or human-controlled vehicles.

1.1 Social Context

Departing from the initial desire to use robots to perform tasks that humans find too dull, too dirty, or too dangerous (also known as the 3Ds) – which mostly affected the manufacturing industry and agriculture – the service sector as well is starting to change due to automation.

This increase in the participation of robots in workplaces is due at least in part to recent breakthroughs in AI and manufacturing, which are starting to enable robots to perform more complicated tasks (soon we may talk about the 4Ds). As a natural consequence, there has been an increase in public debate about robotics, AI, and automation, with particular emphasis on the future of work. Putting aside the influence of sci-fi dystopia evolving robots, public concern on this subject is founded in very objective findings and projections of how society and people's jobs will be affected by this growth in automation.

In that respect, this research should be seen as one step further in the direction of increased automation in society. As the problem of motion planning for multi-robot systems

is discussed, and solutions are proposed, we are actively tilting the scales towards a society where robots are more common and helping to actualize some of the estimates about the future of work.

Many different predictions about the impact of automation can be made. The one that we find most relevant and that can provide some guidelines on how to prepare society to the upcoming changes can be supported by two key figures: the shares of jobs at (high) risk of automation and the amount of training received by low-skilled workers. We say those figures are interesting because, in general, jobs that require low-skilled workers are the first to be automated. Furthermore, low-skilled workers are those that receive the least annual job-related training, making them more susceptible to be left behind by the fast-paced automation.

An OECD (Organization for Economic Co-operation and Development) study [60] from 2018 presents some estimates for those figures for around 30 countries. Fig. 1.1 and 1.2 summarize their findings.

According to that study, around 14% of all jobs among the surveyed countries have a high risk of automation (> 70% risk), with an additional 32% of jobs with risk of significant changes (between 50 – 70%). Furthermore, about 40% of all interviewed workers participate in some job-related training in a 12-month period. That training, however, amounts to merely a few hours per year. If taking only the low-skilled workers into account, that participation in work-related training drops to around 17% for the same period [60].

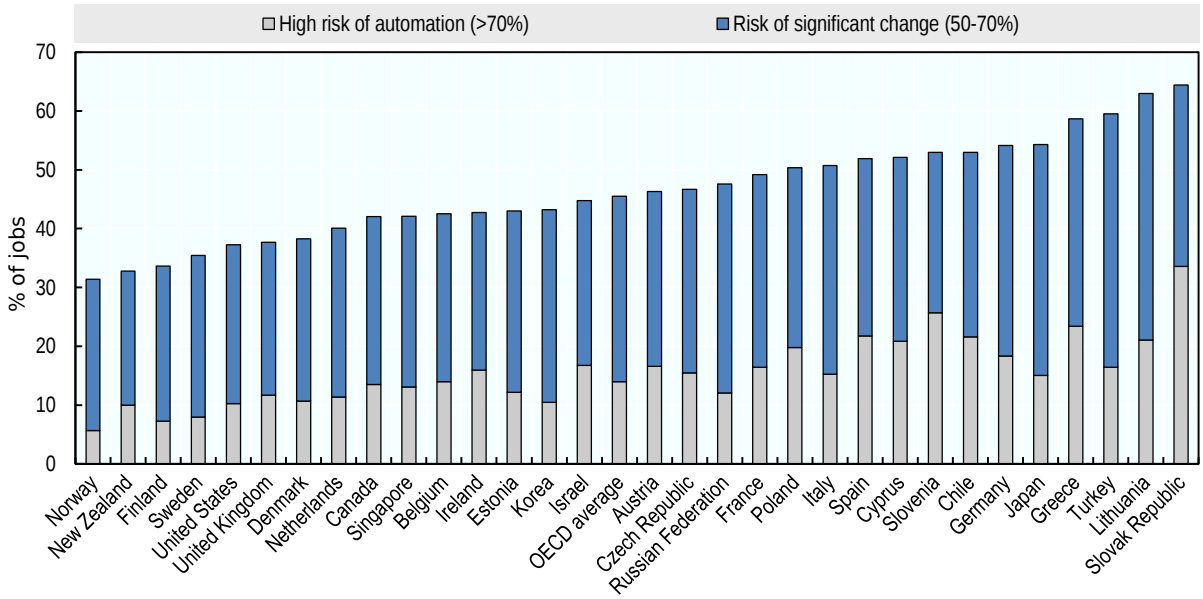


Fig. 1.1 Shares of jobs at risk of automation
Source: OECD (2018), Survey of Adult Skills (PIAAC) 2012, 2015

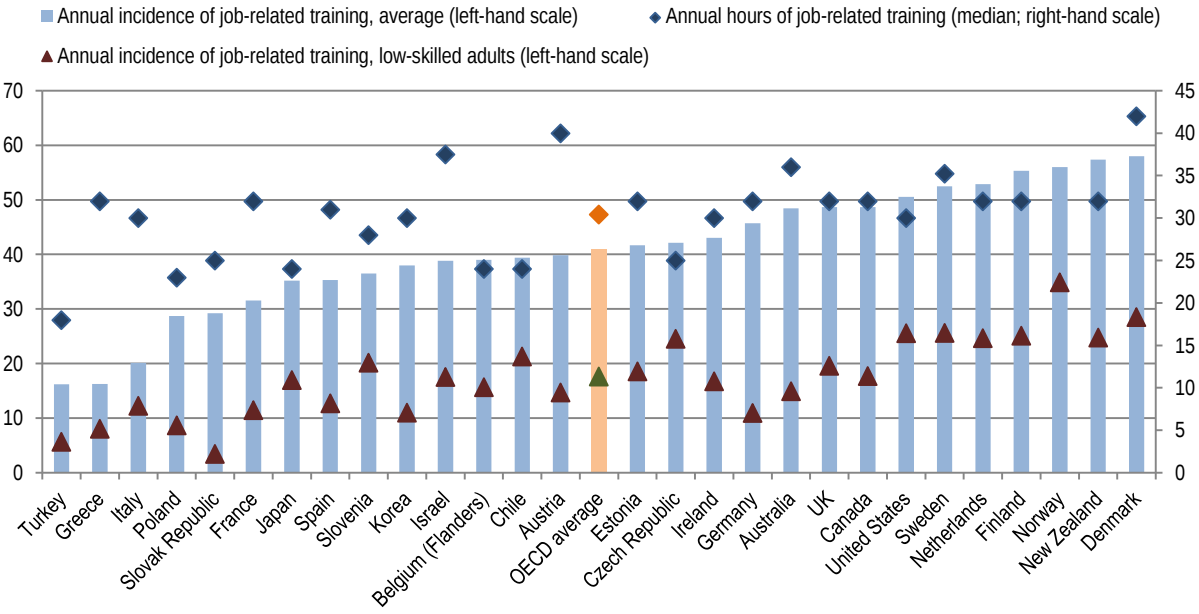


Fig. 1.2 Participation in job-related training undertaken in the 12 months prior to interviews
Source: OECD (2018), Survey of Adult Skills (PIAAC) 2012, 2015

What this indicates is that independently from the overall number of jobs, total earnings, and unemployment rates – which will surely change with increased automation – there is a reasonable chance that the vulnerability of workers occupying positions at risk of automation may be exacerbated with time. The conclusion one may draw is that if we are to prevent a gloomy scenario for the future of work, changes in education and governmental policies concerning mandatory employee training inside companies and organizations have to be made. Adult learning opportunities have to be widely promoted in particular among low-skilled workers as a social response to technological evolution.

From a research and engineering perspective, studies on how to create more cooperative autonomous machines (easier to deploy and to work with) may reduce the need for extensive worker training. In this regard, our research concern with motion planning in dynamic environments and in particular, in the presence of humans aims towards more natural cooperation between humans and mobile robots in a shared space, hopefully attenuating the vulnerability of low-skilled workers.

1.2 Economic Context

The 2020 Multi-Annual Roadmap for Robotics in Europe [79] defines seven domains that capture all parts of the market for robotics technology. Those domains are Manufacturing, Healthcare, Agriculture, Consumer, Civil, Commercial, and Logistics & Transport (see Fig. 1.3

for a summary of this classification). Both laboratories at CEA and ENSTA Paris have research projects closely related to the Logistics & Transport domain.

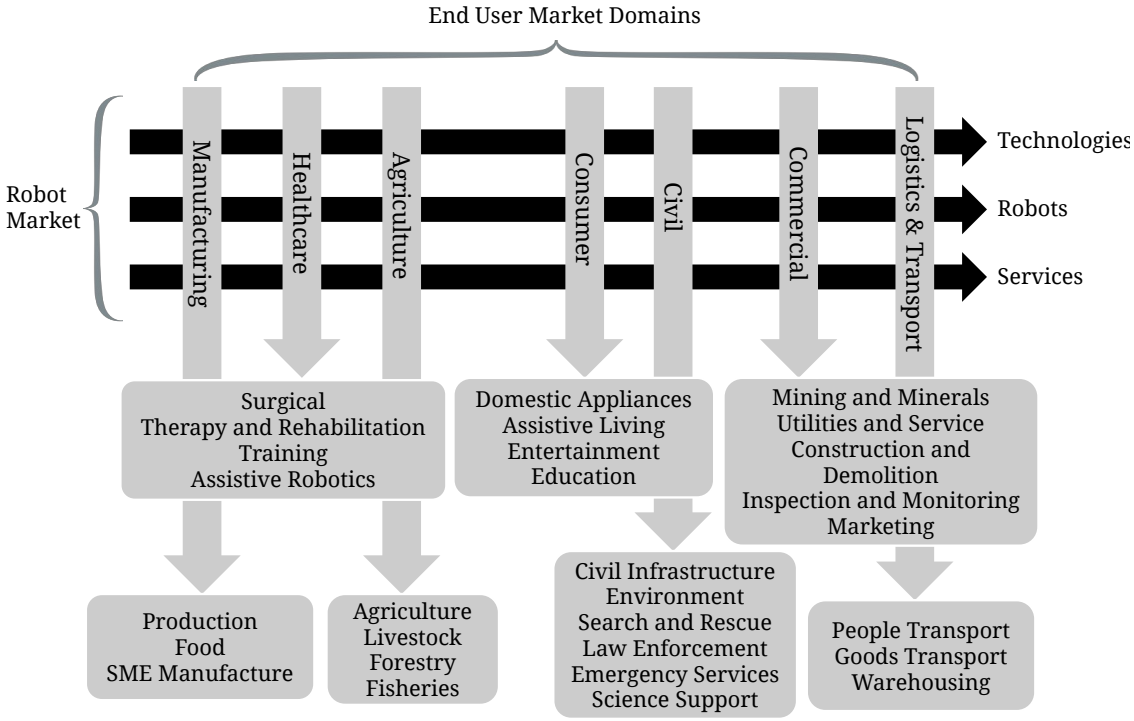


Fig. 1.3 Seven domains encompassing the whole market for robots
Source: Multi-Annual Roadmap For Robotics in Europe Horizon 2020

CEA's projects, such as the fully automated garage for buses [72] and the RATP Group experiment meant to deploy driverless shuttles for people's transportation in a semi-open environment, are examples of recent works in that domain [71]. Similarly, research projects conducted at ENSTA Paris in partnership with the automobile manufacturer Renault study autonomous vehicle path planning in dynamic environments. A representative project among those is the Paris-Saclay Autonomous Lab [73].

In the particular sub-domain of Logistics, the use of robots has experienced rapid growth in past years. According to the "Gartner Supply Chain Technology User Wants and Needs Survey" of 2019 [27], 17% of respondents are already operating mobile robots on their companies. That survey estimates that by 2023, 30% of warehouses workers will be supplemented with collaborative robots. Two primary drivers behind such growth are 1) the need for a reduction in cost in the supply chain (which is inherent to our current global economy based on profit) and 2) regulatory increase in safety and comfort for workers.

Companies such as Amazon and the IDEA Group employ mobile multi-robot systems for autonomously processing client orders for a few years now [29]. This trend is only rising as

indicated by the survey mentioned before, with other companies such as Clearpath, Exotec, iFollow, MiR, Waypoint, and many others, all of which offer robotic solutions for warehouses. The multitude of companies indicates as well the vastness of this still untapped market. Some of those companies already offer systems with a relatively high capacity of coexisting and collaborating with human workers.

This thesis on mobile multi-robot navigation may have a direct use in improving operations related to warehousing such as order processing and optimizing product storage and distribution, while at the same time, improving safety parameters for workers.

1.3 Scientific Context

This work objective is to enable a robotic system composed of multiple mobile platforms (a mobile multi-robot system) to share the workspace among themselves and with workers by performing efficient and secure planning and control of robot movements.

A mobile platform is a system whose ability to change state, its ability to support efforts, and its available energy are limited. Constraints are therefore necessary for generating trajectories, limiting certain transitions between two configurations. This problem is referred to by different terms; the most common ones being *motion planning*, *trajectory planning*, and *kinodynamic planning*.

Depending on the author, those terms may carry some information about the types of constraints that are most relevant. In any case, the consideration of kinematic and dynamic constraints of the real vehicle, such as compliance with speed and acceleration limits, will be necessary for the generated sequence of movements to be achievable by the robotic system. Furthermore, if aiming for completeness and optimality one must also take into account constraints related to geometry, known fixed and moving elements of the environment, and uncertainties derived ultimately from the confidence on sensors and actuators (e.g., unknown areas not covered by sensors, sensors data inherent noise, communication delays).

Due to the complexity of such a class of problems[1], most approaches for solving it tend to use multiple levels of planning. Fig. 1.4 attempts to represent a generic setup going from a high-level mission or task planning down to the physical system to be controlled.

At higher levels of planning, the time-scale is typically in the order of minutes or hours. In the case of a multi-robot system, that part is usually a centralized planner that gathers information from all individual systems and performs task allocation using a simplified

[1] the motion planning problem was shown to be PSPACE-hard [31] for static known environments and NP-hard [15] for dynamic partially known ones

model of the world dynamics. In the context of a warehouse, for instance, this is often closely integrated with the Warehouse Management System (WMS).

At the system level (boxes inside the *Onboard Computation* area) a two-degree of freedom architecture is represented. Its design consists of a trajectory planner and a feedback controller. The trajectory generator provides a feasible feed-forward reference trajectory that satisfies the current set of constraints over a time-scale typically in the order seconds. A feedback controller then stabilizes the system around the reference trajectory. The advantage of this approach is that the system is tracking a feasible trajectory along which it can be stabilized.

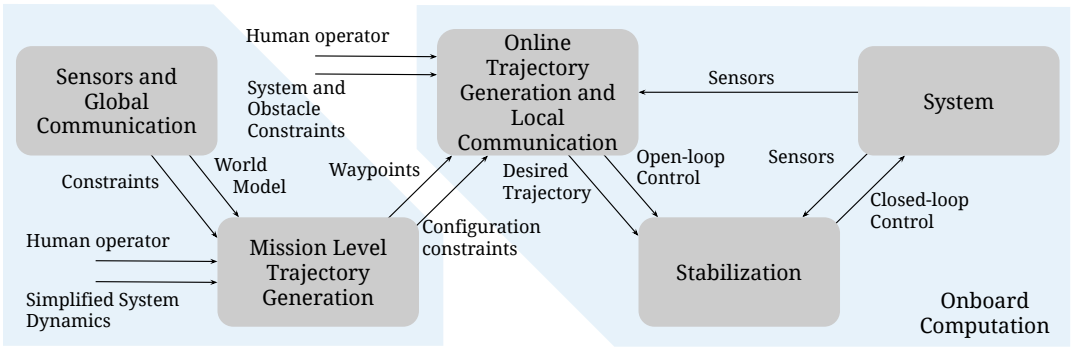


Fig. 1.4 Planning levels

Properties of Motion Planning Approaches

A common way of classifying planning approaches is with respect to the following three properties:

- Quality of the output, or optimality;
- Computational complexity;
- Completeness[2].

Those three properties tend to trade-off with each other, and striving for an algorithm that performs well with regard to all three is challenging.

In addition to those properties, it is common to see two other terms: online planning and real-time planning:

- Online planning refers to planning that interleaves sensing, computation of a plan, and action. It is the opposite of offline planning where the plan is computed at an early moment, before any action, using the currently available information;

[2] to be *complete* or to have *completeness* throughout this thesis means *a search algorithm that is guaranteed to find a solution if one exists*

- Real-time planning is usually a measure of how frequently planning and replanning can be performed. We may say that a planning algorithm is real-time if it can run at a frequency sufficiently high for the assumption of a static environment during one iteration to be reasonable.

We avoid, nevertheless, the overuse of those terms in this work since they can be somewhat vague. In fact, they are better understood as emergent properties derived from the algorithm's computational complexity, size of the problem to be solved (number of constrains and size of the search space), performance of the computers where they run, and implementation techniques (e.g., exploration of concurrency and sparseness properties). Varying those base properties may transform an offline algorithm into an online one. The same goes for real-time planning.

An extra level of classification (and arguably a whole field of research) appears when attempting to plan motions for a multi-robot systems (MRS). One frequent and important way of classifying approaches to the multi-robot motion planning problem (MMP) is into *centralized* and *distributed* (or decoupled).

Centralized approaches are formulated by considering the fleet of robots as one composite robot, meaning they search a solution in the composite configuration space of the whole MRS. They usually provide more guarantees regarding completeness and optimality compared to the distributed approaches. On the other hand, computation time, security vulnerability, and communication requirements can make centralized approaches impracticable, especially for a significant number of robots [12].

For further information on the general state of the art on MMP readers may refer to the comprehensive surveys [32, 59]. In our work, we focus on a distributed approach for our local trajectory planning and feedback control, and a centralized one for the task planning level.

1.4 Contributions and Thesis Outline

To accomplish this thesis objective, we worked on the three central levels shown in Fig. 1.4. Each one is detailed in the following chapters in a middle-out fashion going from our contributions on the feed-forward local trajectory planning, then to our feedback controller, and finally to our work on integrated task and motion planning.

To be more specific, our work proposes:

- An online multi-robot trajectory planning algorithm based on mathematical programming that solves the problem as a constrained optimization problem, presented in Chapter 2;

- A practical implementation of these ideas in a decentralized multi-robot system taking into account real communication delays, also presented in Chapter 2;
- A new Nonlinear Continuous Generalized Predictive Control (NCGPC) law taking into account the output of the previous algorithm for unicycle vehicles in order to improve the quality of the trajectory tracking that will be presented in Chapter 3;
- First steps toward the integration of task and motion planning in partially known environments by exploiting a hierarchical planning approach to decompose the global planning problem into problems solvable by the previous approach, discussed in Chapter 4.

The Chapter 5 summarizes our results and propose some perspectives for future work based on our developments.

These contributions have been published in 3 articles:

- Mendes Filho, J. M. and Lucet, E. (2016). Multi-robot motion planning: a modified receding horizon approach for reaching goal states. Acta Polytechnica, 56(1):10–17 [53];
- Mendes Filho, J. M., Lucet, E., and Filliat, D. (2017). Real-Time Distributed Receding Horizon Motion Planning and Control for Mobile Multi-Robot Dynamic Systems. In ICRA2017 - IEEE International Conference on Robotics and Automation [54];
- Mendes Filho, J. M., Lucet, E., and Filliat, D. (2018). Experimental Validation of a Multirobot Distributed Receding Horizon Motion Planning Approach. In ICARCV 2018 - 15th International Conference on Control, Automation, Robotics and Vision [56].

Chapter 2

Mathematical Programming Approach to Motion Planning

Under the name of *motion planning* (MP), we find two closely-related but different types of problems. When trying to answer the question of how to make a system go from one configuration to another, one may simply be interested in finding a continuous curve in the configuration space[1] connecting initial and goal configurations with no reference to the time variable, i.e., no reference to velocities when moving along that curve or *path*. Another way of addressing the problem is to try to find a *trajectory* – a time-parameterized curve in the configuration space – that satisfies the equations of motion of the system in question. The former formulation is usually called a *path planning problem*, whereas the latter a *complete motion planning problem* (or trajectory planning problem). The line separating the two approaches becomes hazy when one considers the possibility of *decoupled trajectory planning* where first a path is devised and then the problem of finding a feasible trajectory that remains sufficiently close to the path is dealt with.

We argue nevertheless that efficiently planning the motion of a multi-robot system (MRS) requires finding trajectories rather than paths (or at least finding trajectories in addition to paths) – even more so than for single robots. Simply generating paths that guarantee the non-collision of the robots without taking time into account would imply paths that would not, at any point, share conflicting configurations in the space. In reality, non-conflicting trajectories can perfectly well share conflicting configurations as long as those configurations are sufficiently distant from each other in time. Therefore, to hope to achieve optimal coordination of the robots' movements, one should be interested in direct trajectory planning.

[1] a high-dimensional vector space where each possible configuration of a system represents a single point

2.1 Trajectory Planning Problem Definition

In general, a trajectory is understood as a representation of one or more spatial dimensions as function of time that is solution to the motion equations of some system. It is commonly represented by $q(t)$ whereas the motion equation is typically defined by a set of ordinary differential equations written in explicit form $\dot{x}(t) = f(x(t), u(t), t)$ where x is the state of the system, u is the system's controls and f is the vector differential equation describing either only the kinematics (if the state is considered identical to the configuration $x = q$) or the kinematics and dynamics of the system (if the state is considered as $x = [q^T, \dot{q}^T]^T$).

Given those elements, we can more precisely describe the direct trajectory planning problem as *finding a solution* $(q(t), u(t)),\ t \in [t_0, t_f]$ *to the equation of motion that respects a number of constraints such as collision avoidance with robots and obstacles, communication constraints, actuators limits and that takes the system from initial state* (x_{start}) *at time* t_0 *to the final state* (x_{goal}) *at time* t_f. Moreover, among all possible solutions, one may be interested in finding the one that minimizes some objective function J.

From a control theory perspective, the open-loop, feed-forward generated trajectory provides a reference around which to stabilize the system [58]. Such formulation of the problem leads directly to the field of optimal control, and subsequently, that of mathematical programming explored in this thesis. In the literature, however, we find a large number of other types of approaches as well, which we will quickly review in the next section.

Note that since f do not (ever) perfectly describe the motion of the system – especially if it is only a kinematic model – we will only be interested in using the q part of the solution and leave u to be computed by a lower-level controller. Such an approach also raises the matter of computational time given that control inputs must be sent as quickly as possible and solving the MP problem as an optimization problem with several constraints can take a relatively long time.

2.2 State of the Art on Motion Planning

We are particularly interested in solving the problem of planning trajectories for a team of nonholonomic mobile robots, in a partially known environment occupied by static obstacles, being efficient with respect to the travel time (amount of time to go from initial to goal configuration).

The first textbook on the subject of coordinated motion planning of multiple robots [43] was published a bit less than three decades ago. After that first book, several other cover this topic in details, such as [44, 17, 45]. Meanwhile, a considerable amount of scientific

articles on the subject of motion planning for MRS was published in the past years. Through the rest of this section, we will quickly review the related work to this subject going from least specific (broad families of motion planning in general) to more specific (Mathematical Programming approaches to MRS motion planning).

2.2.1 General Motion Planning Approaches

As explained at the beginning of this chapter, motion planning encompasses the problem of path planning (spatial planning), trajectory planning (spatio-temporal planning), and all hybrid or in-between approaches. The approaches to MP can be organized in a few groups of methods: Configuration Space Discretization, Potential Field, Elastic Band, Dynamic Window, Velocity Obstacles, and Mathematical Programming.

Configuration Space Discretization Methods

These methods aim at capturing the connectivity of the configuration space (C-space) into a graph and then perform a graph search to find a path from initial to final configuration. They can be further split into Roadmaps or Sampling-Based groups of algorithms.

Roadmaps Methods rely on the assumption that a structured C-space is available with free (C_{free}) and occupied (C_{obst}) C-spaces defined. They can employ different techniques for discretizing the C_{free}, the main ones being:

- Visibility graphs [7]: requires the forbidden part of the C-space to be described as polygons and creates a roadmap by connecting all vertices of those set of polygons;
- Voronoi diagrams [82]: has the same requirement as the Visibility graphs technique but produces a roadmap with the maximum clearance from all obstacles;
- Exact cell decomposition [35]: decomposes C-space into non-overlapping cells of varying shapes and then constructs connectivity graph to represent adjacencies;
- Approximate cell decomposition [6]: same as exact cell decomposition but uses a fixed predefined shape for creating the cells.

A variant called Reactive Deformation Roadmaps (RDRs) introduced in [28] extends the roadmaps method for multi-robot motion planning by using deformable links and dynamically retraction to capture the changing connectivity of the free space. To the best of this author's knowledge, this method was only tested in simulation.

Roadmaps methods have the advantage of being complete[2] but can quickly become intractable. Sampling-Based Algorithms attempts to avoid this shortcoming by trading-off completeness against efficiency.

[2] with respect to the discretized C-space

Sampling-Based Algorithms depart from the idea of directly characterizing C_{free} and C_{obst} and rely on a collision detection evaluation to judge if some configuration lies on the C_{free}, which is usually a computationally cheap operation.

Initial approaches of this type, such as Probabilistic Roadmaps Planner [34] was done using a uniform random distribution for sampling the C-space. Faster algorithms suited for specific applications and even involving kinematic and dynamic constraints were created, such as RRT or RRT^X [64].

Historically, sampling-based algorithms for a single robot have been extended to the multi-robot case by mainly centralized approaches [76]. Distributed sampling-based techniques for MMP exist but they are usually limited to discrete domains. Many perform decoupled path planning first, and then search velocity profiles that avoid collisions between robots, which is not a complete approach [78].

Given any of the discretization above, a search algorithm has to be used to find a path in the graph-like description of the configuration space. Dijkstra's algorithm, A*, Any Angle A*, D*, D* Lite are widely used[3].

Potential Field Methods

Initially proposed in [37], this type of approach models the vehicle as a particle under the influence of an artificial potential field created from repulsive forces from obstacles and attractive force from goals. The C-space is typically discretized as fixed-size grids and the vector field is defined as a value associated with each cell. Costmaps created from occupancy grids where an inflation method attributes values to each cell as a function of the distance to the closest obstacle are another way of applying this same idea.

The robot controller then follows the gradient of the potential field in order to avoid obstacle and reach the goal.

Elastic band

First presented in [70], this approach attempts to fill the gap between path planning and control. This algorithm takes an already computed path (using, for instance, one of the methods mentioned before) and model that path as a deforming collision-free path or elastic band. They use the so-called *bubbles* to construct local subsets of C_{free} and to minimized the deforming force on the elastic band that yields a collision-free trajectory.

Dynamic Window

[3] Field D* was used on Mars rovers Spirit and Opportunity [16]

Initially presented in [26], the Dynamic Window Approach (DWA) assumes the existence of a planned path or sequence of waypoints. It constructs a velocity search space from the reachable and admissible velocities of the robot within for time window t discarding illegal velocities (the ones causing collisions). It uses then a heuristic navigation function (an objective function) to search the optimal velocity for controlling the mobile platform.

Velocity Obstacles

Work presented in [22] gives a critical analysis of the most important variants of this family of approaches, namely the classical velocity obstacle (VO), reciprocal velocity obstacle (RVO), hybrid-reciprocal velocity obstacle (HRVO) and optimal reciprocal collision avoidance (ORCA) approaches. This family of motion planning methods was first introduced in [25] and its main idea is to perform a search of the reference control input in the velocity space, based on the current positions and velocities of the robots and obstacles.

Mathematical Programming

In its most general form, a mathematical program minimizes (or maximizes) an objective function subject to a set of constraints over continuous and discrete decision variables. It offers flexibility to explicitly and simultaneously accommodate multiple systems requirements. In most cases, these requirements are a subset of the following list: kinematics, dynamics, collision avoidance, and connectivity maintenance requirements [2].

An increasingly common technique associated with Mathematical Programming for MRS motion planning is the Receding Horizon Approach, mainly inspired by works in Distributed MPC [49]. The approach employed in our work and detailed in the rest of this chapter fits into this category of mathematical programming approaches.

2.2.2 Mathematical Programming Approaches to Multi-robot Motion Planning

Through the rest of this section we will consider recent works that closely relate to ours, all of them being distributed, with an emphasis on mathematical programming approaches.

Works presented in [20, 87, 86] formulate the trajectory planning for an MRS as an optimization problem and use the distributed receding horizon approach for coping with the dynamic environment and disturbances.

In [20, 87] each robot optimizes only its own trajectory at each computation/update horizon. In order to avoid robot-to-robot collisions, neighbor robots exchange information before performing an update. In [20] initial intended trajectories are computed by each

robot ignoring constraints that take the other robots into account, and these trajectories are then exchanged with the neighbor robots. Similarly, in [87], robots are required to communicate and exchange their current and most recent states; however, that exchanged information is used to predict robots' trajectories assuming uniform linear motion. Then, those predictions or intended trajectories are used to form collision avoidance constraints in the optimization problem.

An interesting aspect of the approach in [87] is that it splits the planning horizon into two parts: during the first part, collision avoidance and smoothness of trajectories are dealt with; in the second part, only global target convergence is a concern. An incremental sequential convex programming (iSCP) algorithm for solving the optimization problems is used. Only results in simulations are shown. In [20], an SQP (Sequential Quadratic Programming) algorithm is used for solving the optimization problem.

Differing from the two previous approaches, in [86], the MP is formulated as a single global optimization problem and then decoupled and distributed among the robots using the Alternating Direction Method of Multipliers (ADMM) [13]. In order to reduce the update time and communication between the agents, an approach is proposed for which only one ADMM iteration is performed per trajectory update. The approach requires a detailed geometric description of the environment, but in those conditions, it was shown to work in simulation and with real holonomic vehicles (extension to nonholonomic is said to be possible).

Work presented in [24] proposes a Decentralized Nonlinear Model Predictive Control (DNMPC) for addressing MRS MP where careful convergence and feasibility analysis are provided. Formation maintenance, avoidance of static obstacles, and inter-robot collision avoidance are verified in simulation for unicycle robots, but the approach could be generalized for other types of systems. The method requires, though, that the robots communicate sequentially and it is not clear how the underlying finite horizon optimization control problems for each robot are solved. This approach as well remains to be tested in a real experiment.

Another recent relevant work on Distributed MPC for MRS is presented in [52]. Instead of relying on complete predictions of other robots' trajectories, it uses occupancy grid data aiming for a reduction in the required communication means. The approach was tested on nonholonomic mobile platforms using an external motion capture system for providing a ground truth localization of the mobile platforms.

Our approach closely relates to those presented in [20, 87, 86], in particular [20] from which we based our developments. Some of [20] identified drawbacks are its dependence on several parameters for achieving real-time performance and good solution optimality,

the difficulty to adapt it for handling dynamic obstacles, the impossibility of bringing the robots to a precise goal state, the limited geometric representation of obstacles, and the fixed values used for localization and tracking errors.

Some of those drawbacks are shared by the work in [87, 86]. The main differences from our work to theirs consist in how collision constraints are handled and how localization and tracking errors are modeled. In [87], the problem of having non-differentiable constraints for obstacle avoidance is addressed by transforming them into smooth nonlinear constraints. Conversely, our work derives differentiable smooth constraints from sampled data (inflated occupancy grids given by the perception module) by doing local interpolations around sampled points in robots' planned trajectories. As for localization and tracking errors, [87, 86] assume they are always inferior to a small constant while our work uses a probabilistic model and confidence regions to produce robust collision-free trajectories.

2.3 Mathematical Programming General Formulation as Nonlinear Optimization Problem

We will now introduce the general formalization that is the support of our optimization-based trajectory planning approach. The optimization problem, as stated below, gives a general formulation for the trajectory planning problem described in section 2.1:

$$\underset{(t_f, q(t), u(t))}{\text{argmin}} \quad J(t_f, q(t), u(t)) \tag{2.1}$$

$$\text{subject to} \quad \dot{x}(t) = f(x(t), u(t), t), \qquad t \in [0, t_f] \tag{2.2}$$

$$u_{\min}(x(t)) \leq u(t) \leq u_{\max}(x(t)), \qquad t \in [0, t_f] \tag{2.3}$$

$$h(q(t)) \leq 0, \qquad t \in [0, t_f] \tag{2.4}$$

$$x(0) = x_{\text{start}}, \quad \dot{x}(0) = \dot{x}_{\text{start}} \tag{2.5}$$

$$x(t_f) = x_{\text{goal}}, \quad \dot{x}(t_f) = \dot{x}_{\text{goal}} \tag{2.6}$$

where J (Eq. (2.1)) represent the cost to be optimized, inequation (2.4) represents constraints on configurations, typically due to obstacle avoidance; $u_{\max}$ and $u_{\min}$ represent limitations on the system's actuators; without loss of generality, we assumed $t_0 = 0$ in order to simplify notations.

Note that a team of robots could be treated as a single robotic system in this formulation. Trajectory planning would then be done in a centralized fashion where q would represent the aggregated configuration vector of all robots in the system. However, this approach is often

impractical because of scalability and robustness issues; centralized computation typically represents a single point of failure for the MRS, and computational and communication resources may be insufficient for more than a few robots. Indeed, many algorithms for numerically solving optimization problems have a strong computational cost dependency on the number of parameters (see annex A.1 on the numeric solving algorithms), and therefore solving for multiple robots together becomes quickly prohibitive.

Here, we are going to consider the above optimization problem to concern only individual robots in a system. Coupling constraints allowing the robots to coordinate their trajectories with those of other robots in the system are going to be added later to the optimization problem. For now, however, they can be considered as being part of the inequality constraints in (2.4).

The set of methods for numerically solving such optimization problems are often named *Transcription Techniques*. These methods work by converting a continuous problem (equations (2.1) to (2.6)) into a non-linear programming problem that can be numerically solved by standard algorithms. Two broad classes of transcription techniques exist: shooting methods and simultaneous methods. For the purpose of directly computing the trajectory q, only the latter class is relevant. Shooting methods do not directly represent the state trajectory in the decision variables of the optimization problem, only the control input relying then on simulation to enforce the system dynamics. For a comprehensive explanation on the subject see [36].

In order to apply the class of simultaneous methods for transcription, the following approximations are needed:

- Continuous-time constraints need to be approximated by a finite number of constraints. This is usually done by uniformly sampling inside the time interval $[0, t_f]$;
- A finite-parameter representation of the solution needs to be used. That representation will define the decision variables of the optimization problem. That can be done either by:
  - Parameterizing the trajectory $q(t)$ and finding solutions for $u(t)$ using (2.2);
  - Parameterizing the controls $u(t)$ and finding solutions for $q(t)$ by integrating (2.2);
  - Parameterizing both and evaluating (2.2) at a fixed number of points over the interval $[0, t_f]$ to ensure that this constraint is respected.

The second method requires integrating the motion equation, whereas the third adds more parameters and constraints for which to solve the optimization problem. The first method, although more promising concerning the number of computations, is not straight

forward either. There is no unicity of parametric representation of a given trajectory as pointed out in [68], and choosing one depends on its properties and the problem's nature.

The choice of a parametric representation of the trajectory will have a significant impact on the stability and efficiency of the numerical optimization [17]. Any possible parametric representation of a curve will have very well defined properties that may or may not be suitable for representing the trajectory of a robotic system and for approximating it through optimization numerical solvers. Some few usually desirable properties are listed below:

- local support – changing a parameter changes its local neighborhood while leaving the rest of the trajectory unchanged
- smoothness – the trajectory should be of class C^k with $k \geq 2$ so first and second derivatives can be continuous functions;
- stability – both by avoiding oscillations between interpolated points and by being numerically stable;
- low computational complexity.

Independently of the choice of the type of parameterization it is reasonable to assume a vector Γ of parameters used for approximating the trajectory: $q(t) \simeq \tilde{q}(t,\Gamma)$. To keep the notation simple, the tilde over q will be omitted. Furthermore, assuming that N evaluations spaced by $T = t_f/(N-1)$ intervals are performed for each continuous-time constraint equation, we may reformulate the NLP as follows:

$$\underset{(t_f,\Gamma)}{\text{argmin}} \quad J(t_f,\Gamma) \tag{2.7}$$

$$\text{subject to} \quad u_{\min}(kT,\Gamma) \leq u(kT,\Gamma) \leq u_{\max}(kT,\Gamma), \qquad k = 0 \ldots N-1 \tag{2.8}$$

$$h(q(kT,\Gamma)) \leq 0, \qquad k = 0 \ldots N-1 \tag{2.9}$$

$$q(0,\Gamma) = q_{\text{start}}, \quad \dot{q}(0,\Gamma) = \dot{q}_{\text{start}} \tag{2.10}$$

$$q(t_f,\Gamma) = q_{\text{goal}}, \quad \dot{q}(t_f,\Gamma) = \dot{q}_{\text{goal}} \tag{2.11}$$

Another important matter regarding the optimization problem as defined above is its temporal scope vis-à-vis the nature of the work environment of the autonomous system. In a known static environment, planning could be done globally for the whole task of going from q_{start} to q_{goal} and executed afterward. However, in an unknown, dynamic environment, solutions produced by global planning may become outdated as the environment evolves through time. This problem leads us to propose a receding horizon approach for planning described in the next section.

2.4 Formulation for a Partially Known Environment

As a team of robots evolves in their work environment, they progressively perceive new obstacles in their way to their goal configuration. Hence, trying to plan for the whole motion from initial to goal configurations is not a satisfying approach. Planning locally and replanning is more suitable for taking new information into account as it comes.

Naturally, planning locally leaves some multi-robot coordination problems unsolved, such as deadlocks and livelocks. The approach of this thesis is to strive for a good compromise between optimality, robustness, and feasibility at the trajectory planning level and assume that other, tightly integrated layers of planning will take care of the remaining problems. Chapter 4 on task planning will present this thesis work in that direction where part of the proposed approach relies on efficient global path planning algorithms (namely A*) for solving the more broad motion planning problem.

One possible way to keep the mathematical programming framework for motion planning in a partially known, dynamic environment is to use a receding horizon approach: planning will be performed online, locally, for finite time windows that slide forward as the robot evolves in its workspace. This approach will provide quickly computed plans that are partially executed and recomputed periodically to take the environment evolution into account.

2.4.1 Interleaving Planning, Perception, and Execution by a Receding Horizon Approach

The principle of the receding horizon approach is to have a prediction time-horizon T_p and an implementation/computation timeslot T_c with $T_c \ll T_p$. T_p is the time-horizon for which a local solution to the motion problem will be created, and T_c is the timeslot during which a portion of that solution is executed while the next trajectory - created for the next time-horizon T_p - is being computed (see figure 2.1). It differs from the classical definition of MPC since not only the first value of the optimized solution is used for computing the system's input.

Using the trajectory computation framework presented previously, the optimization problem at step $n \mid n \in \mathbb{N}_0$ (from here on denoted NLP_n) becomes:

$$\underset{(\Gamma_n)}{\text{argmin}} \quad J(\Gamma_n) \tag{2.12}$$

$$\text{subject to} \quad u_{\min}(kT + nT_c, \Gamma_n) \leq u(kT + nT_c, \Gamma_n) \leq u_{\max}(kT + nT_c, \Gamma_n), \quad k = 0 \ldots N-1 \tag{2.13}$$

$$h(q(kT + nT_c, \Gamma_n)) \leq 0, \quad k = 0 \ldots N-1 \tag{2.14}$$

$$q(nT_c, \Gamma_n) = q_{\text{prev}}, \quad \dot{q}(nT_c, \Gamma_n) = \dot{q}_{\text{prev}} \tag{2.15}$$

q_{prev} and $\dot{q}_{\text{prev}}$ are defined as follows:

$$q_{\text{prev}} = \begin{cases} q_{\text{start}}, & \text{if } n = 0 \\ q(nT_c, \Gamma_{n-1}), & \text{otherwise} \end{cases} \tag{2.16}$$

$$\dot{q}_{\text{prev}} = \begin{cases} \dot{q}_{\text{start}}, & \text{if } n = 0 \\ \dot{q}(nT_c, \Gamma_{n-1}), & \text{otherwise} \end{cases} \tag{2.17}$$

Since the time scope of the optimization problem changes from $[0, t_f]$ to $[nT_c, nT_c + T_p]$, T represents $T_p/(N-1)$. See figure 2.1 for a visual representation of how plans would occur with time.

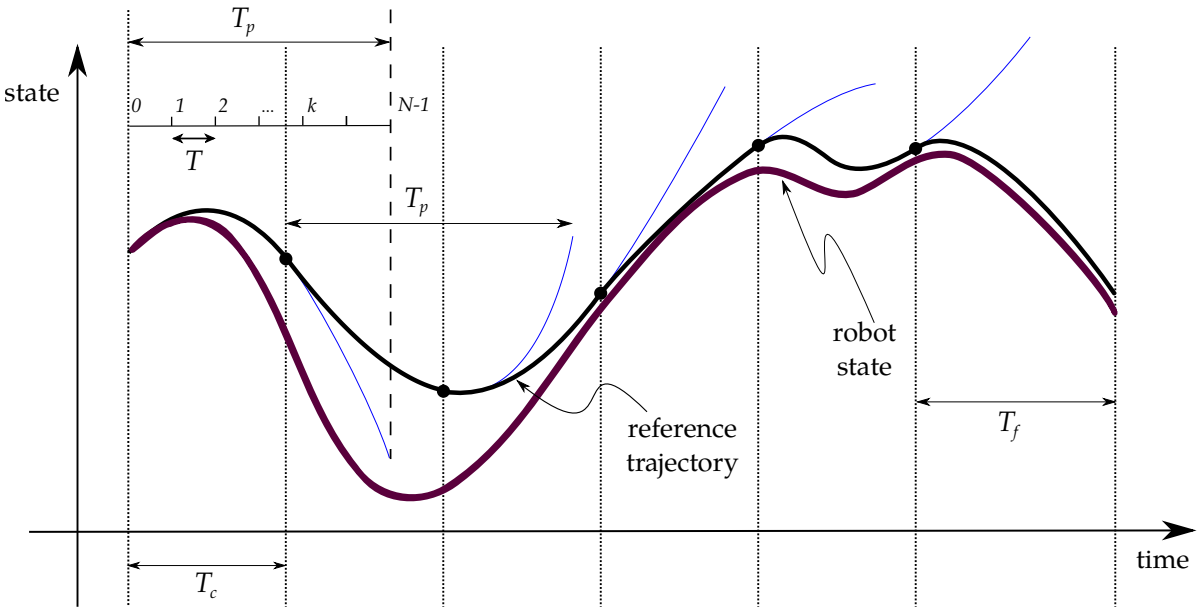


Fig. 2.1 Receding Horizon Scheme

Before each n-th search for a solution to the above problem, an update step is performed where the latest available information about the robot's environment is taken into account.

In the above formulation one may notice that q_{goal} and $\dot{q}_{\text{goal}}$ are not present. To enable the robots to reach their goal states, the receding horizon approach has to stop near the vicinity of the goal state and a different approach has to be used.

2.4.2 The Termination Problem

After stopping the receding horizon planning algorithm, we propose a termination planning that considers those constraints related to the goal state. The detailed definition of the resulting optimization problem is presented further in this chapter at subsection 2.6.5.

The criterion used to pass from the receding horizon planning to the termination planning is based on the distance between goal and current position of the robots d_{rem}. It is defined by the equation:

$$d_{\text{rem}} \leq d_{\text{min}} + v_{\text{max}} T_c \tag{2.18}$$

where v_{max} represents the maximum linear speed of the robot.

This condition ensures that the termination plan will be planned starting from a position at least d_{min} distance from the robot's goal position (see figure 2.2). This minimal distance is arbitrary, but sensibles values may be greater than the breaking distance of the robot from maximum speed at constant maximum acceleration ($v_{\text{max}}^2/(2a_{\text{max}})$) and not much larger then $v_{\text{max}} T_p$.

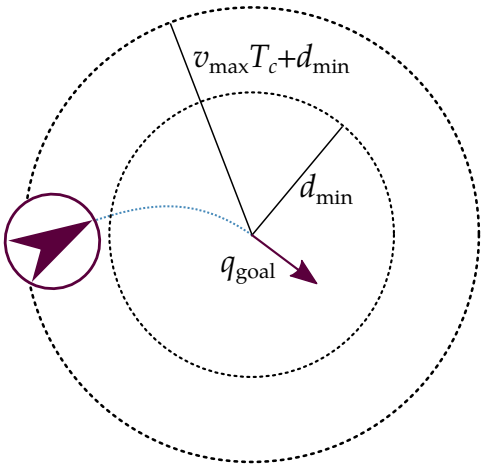


Fig. 2.2 Representation of termination reference trajectory triggered before robot gets closer then d_{min} from its goal

Before solving the termination planning problem, new parameters for trajectory representation and computation are calculated, taking into account the estimated remaining distance and the typical distance traveled during T_p. This is done in order to adapt the configuration intended for a trajectory lasting T_p to one that starts at point A and finishes at point B where norm$(A-B)$ is between $d_{\min}$ and $d_{\min} + v_{\max} T_c$.

2.5 Taking Multiple Robots into Account

In order to take potential conflict between robots' trajectories into account and find solutions that avoid collision between robots, we use a two-step algorithm. In the first step, each robot solves a receding horizon optimization problem to find a trajectory regardless of being part of an MRS. Then, if necessary, a second optimization problem that takes into account conflicting robots' information is solved. This approach is called Distributed Receding Horizon Motion Planning (DRHMP).

For each receding horizon planning problem, the following steps are therefore performed:

Step 1 Each robot in the team computes its own intended solution trajectory (denoted $(\hat{q}(t), \dot{\hat{q}}(t), \ddot{\hat{q}}(t))$[4] with q the configuration vector of the robot) by solving the previously proposed constrained optimization problem. In that optimization problem, all constraints are included except coupling constraints, that is, constraints that involve solving a conflict between multiple robots such as collision or loss of communication.

Step 2 Robots involved in a potential conflict (risk of collision, loss of communication) update their trajectories computed during Step 1 by solving a second constrained optimization problem that additionally takes into account geometric constraints for avoiding conflicts with other robots. This is done by using estimates of the intended trajectories of the other robots. If a robot is not involved in any conflict, Step 2 is not executed and its final solution trajectory is identical to the one found at Step 1.

Differently from [20], we do not consider that all robots involved in a conflict have finished Step 1 and exchanged information when any of them starts executing Step 2. Here, the robot estimates trajectories for the conflictual robots based on the available information at the end of Step 1. Those estimates allow asynchronous communication between robots:

[4] higher order derivatives of $\hat{q}(t)$ have to be guaranteed to exist by the choice of trajectory representation

they can use different T_p and T_c, and no defined frequency for their communication is imposed. Planning proceeds regardless of the communication frequency.

Since Step 1 and Step 2 solve different versions of NLP_n (by varying the meaning of inequation h) their respective optimization problems are denoted $\text{NLP}_{n,1}$ and $\text{NLP}_{n,2}$. Likewise, optimizations problems solved after the receding horizon procedure, at the termination stage, are denoted $\text{tNLP}_{n,1}$ and $\text{tNLP}_{n,2}$ for they have different constraints and objective functions. Details about the actual implementation of these different formulations for specific robots can be found in subsection 2.6.5.

Figure 2.3 present the global flowchart of the complete approach.

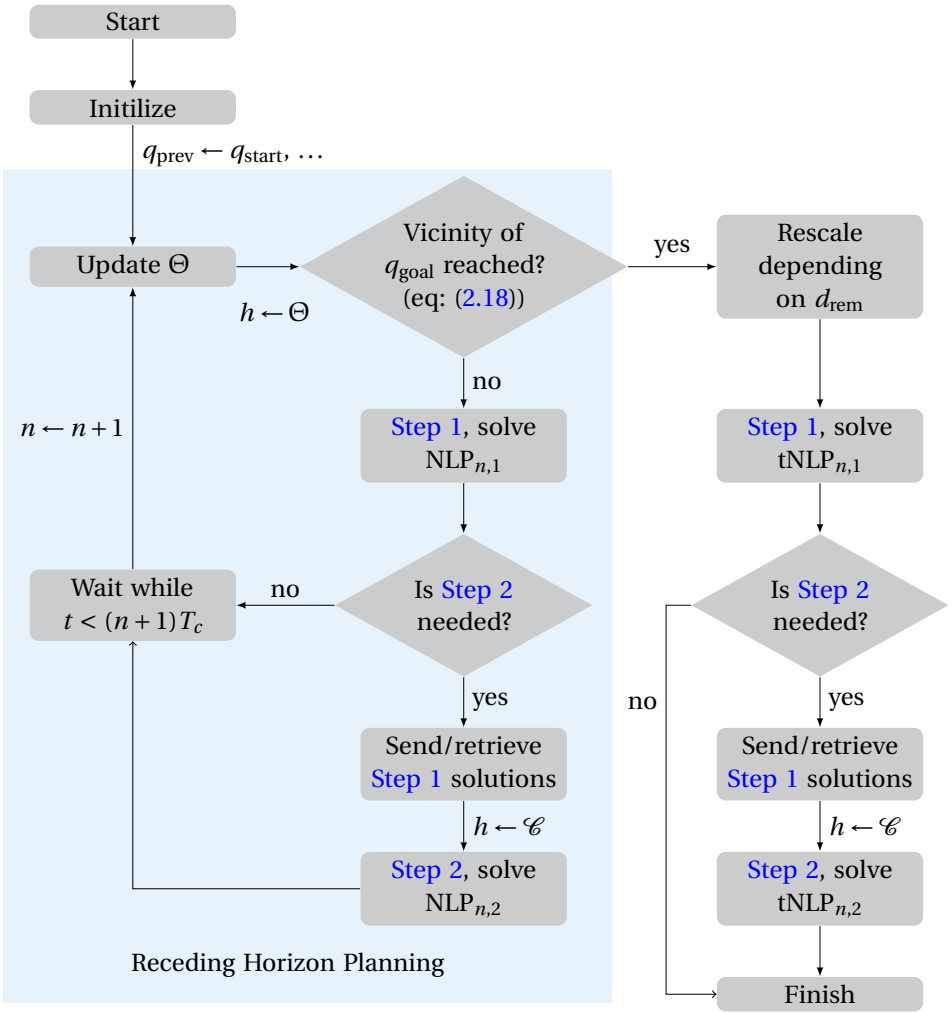


Fig. 2.3 Flowchart illustrating the distributed local motion planning. Θ represents exteroceptive observation (namely obstacles), $\mathscr{C}$ represents communicated information exchanged among robots

2.6 Implementation for Unicycle-like Mobile Vehicles

Before presenting experimental evaluations of the proposed approach, let us first introduce some implementation choices regarding the robotics platform and the trajectory parameterization.

We chose one of the most common types of wheeled mobile robots: differential wheeled robots. They can be modeled by the simple unicycle system:

$$\begin{gathered}\dot{q} = \mathrm{f}(q,u) \Rightarrow \\ \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ w \end{bmatrix}\end{gathered} \tag{2.19}$$

All experimental work was based on this model and its properties. Similar work using other robotic systems such as drones can be found [58].

2.6.1 Model and Flatness Property of the System

Using the flatness property of a system [48], it is possible to be exclusively interested in planning a trajectory for the flat output variable z provided it exists.

For the unicycle model, the flatness property holds and $z = [x,\ y]^T$ is the flat output variable. The equations (2.20) and (2.21) show how the state variables and control variables can be computed from the flat output and its first l^{th} derivatives. Those equations may be used whenever it is needed to retrieve the fundamental variables:

$$\begin{gathered}\varphi_1(z(t_k),\ldots,z^{(l-1)}(t_k)) = \\ \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \arctan(\dot{z}_2/\dot{z}_1) \end{bmatrix}\end{gathered} \tag{2.20}$$

$$\begin{gathered}\varphi_2(z(t_k),\ldots,z^{(l)}(t_k)) = \\ \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \sqrt{\dot{z}_1^2+\dot{z}_2^2} \\ \dfrac{\dot{z}_1\ddot{z}_2-\dot{z}_2\ddot{z}_1}{\dot{z}_1^2+\dot{z}_2^2} \end{bmatrix}\end{gathered} \tag{2.21}$$

$$\begin{gathered}\varphi_3(z(t_k),\ldots,z^{(l+1)}(t_k)) = \\ \begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \frac{\dot{z}_1\ddot{z}_1+\dot{z}_2\ddot{z}_2}{\|\dot{z}\|} \\ \frac{(\ddot{z}_1\ddot{z}_2+z_2^{(3)}\dot{z}_1-(\ddot{z}_2\ddot{z}_1+z_1^{(3)}\dot{z}_2))\|\dot{z}\|^2-2(\dot{z}_1\ddot{z}_2-\dot{z}_2\ddot{z}_1)\|\dot{z}\|\dot{v}}{\|\dot{z}\|^4} \end{bmatrix}\end{gathered} \tag{2.22}$$

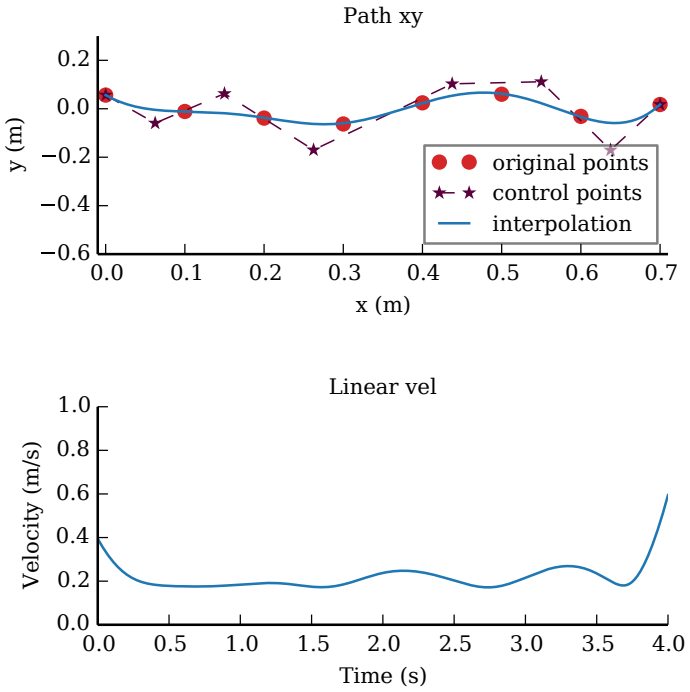


Fig. 2.4 Clamped B-spline curves followed by its first derivative along x (as function of the parametric variable time)

Using this property, it is, therefore, sufficient to plan a $[x,\ y]^T$ trajectory instead of a trajectory in the configuration space. This change of variables makes the trajectory representation simpler.

2.6.2 Trajectory Parameterization

Many different types of trajectory or path representation have been used throughout the history of motion planning. Work presented at [1] contains a good summary of the main different path primitives from straight lines and circular arcs, to complex frameworks dealing with circular arcs, clothoids, and quintic curves at the same time.

In our work, we settled for using B-splines curves (see Fig. 2.4 and their definition in the note below) since they are extensively used in engineering for having advantageous mathematical and algorithmic properties. More specifically, they meet the required properties listed in section 2.3 emphasized here below:

- B-spline algorithms are fast and numerically stable;
- Curves are invariant under common geometric transformations, such as translation and rotation;

- B-spline gives the minimum local support;
- Derivatives are continuous up until its given degree.

We chose a B-spline representation of degree 3 to ensure a continuous second derivative.

Note

Given $n+1$ control points $P_0,\ P_1,\ \ldots,\ P_n$ and a knot vector $U = \{u_0, u_1,\ \ldots, u_m\}$, the B-spline curve of degree p defined by these control points and knot vector U is:

$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i \tag{2.23}$$

where $N_{i,p}(u)$ are B-spline basis functions of degree p.
The i-th B-spline basis function of degree p, written as $N_{i,p}(u)$, is defined recursively as follows:

$$\begin{aligned} N_{i,0}(u) &= \begin{cases} 1 & \text{if } u_i \le u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} \\ N_{i,p}(u) &= \frac{u-u_i}{u_{i+p}-u_i} N_{i,p-1}(u) + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} N_{i+1,p-1}(u) \end{aligned} \tag{2.24}$$

If the knot vector does not have any particular structure, the begin and end of the generated curve will not coincide with the first and last control points. This type of B-spline curves is called open B-spline curves (see Fig. 2.5). We may want to clamp the curve so that it is tangent to the first and the last legs at the first and last control points, respectively, as a Bézier curve does. To do so, the first knot and the last knot must be of multiplicity $p+1$. This condition will generate the so-called clamped B-spline curves (see Fig. 2.6). Implementations of clamped B-splines are more common than open, and, for the sake of convenience, we used the former type for representing robots trajectories in our work. A thorough introduction to B-splines can be found in [10].

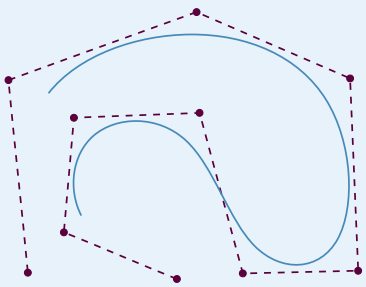


Fig. 2.5 Open B-spline

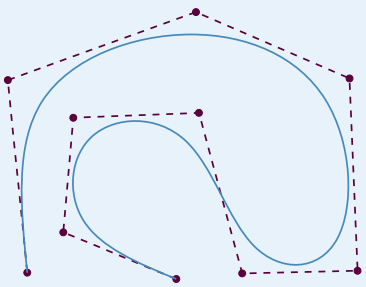


Fig. 2.6 Clamped B-spline

2.6.3 Optimization Algorithm

The need for a solver (or optimizer) that supports nonlinear equality and inequality constraints restricts the number of numerical optimization solvers to be considered.

For our initial implementation of the motion planning algorithm, the SLSQP optimizer stood out as a good option (a review of other numerical optimization algorithms we considered can be found in Appendix A, section A.1). Besides being able to handle nonlinear equality and inequality constraints, its availability in the minimization module of the open-source scientific package Scipy [21] helps to facilitate the motion planner implementation in python. For C++ development, the NLopt project offers the option to use that type of solver as well.

The use of SLSQP is not without its pitfalls, however. An error was experienced using this optimizer based on the SLSQP implemented by Dieter Kraft [38] (the one used by the two projects mentioned before). As the cost function value becomes too high (typically for values greater than 10^3), the optimization algorithm finishes with the "Positive directional derivative for linesearch" error message. This error appears to be a numerical stability problem experienced by other users of the NLopt library.

For working around this problem, we proposed a change in the objective functions of the receding horizon optimization problems. This change aims to keep the evaluated cost of the objective function around a known value when close to the optimal solution instead of having a cost dependent on the goal configuration (which can be arbitrarily distant from the current position).

We simply exchanged the goal position point in the cost function by a new point computed as follows:

$$p_{\text{new}} = \frac{p_{\text{goal}} - p(nT_c, \Gamma_{n-1})}{\text{norm}(p_{\text{goal}} - p(nT_c, \Gamma_{n-1}))} \alpha T_p v_{\max}$$

Where p_{goal} and $p(nT_c, \Gamma_{n-1})$ are the 2D positions associated with configurations q_{goal} and $q(nT_c, \Gamma_{n-1})$ respectively, $\alpha \mid \alpha \geq 1, \alpha \in \mathbb{R}$ is a constant for controlling how far from the current position the new point is placed, the product $T_p v_{\max}$ the maximum possible distance covered by the robot during a planning horizon.

Normalization of constraints was also implemented in order to make the problem well-conditioned.

2.6.4 Environment Representation

Instead of a complete geometric description of obstacles in the environment, sensor data from the mobile robots are typically converted to some discrete representation with portions of space being marked as occupied or free. For instance, under ROS, the `move_base` node (responsible for trajectory planning and control of robots - see Fig. 2.7) uses 2D costmaps. These costmaps are based on occupancy grids generated from 2D or 3D sensor data and user-defined inflation radius that is meant to take the robot's footprint into account.

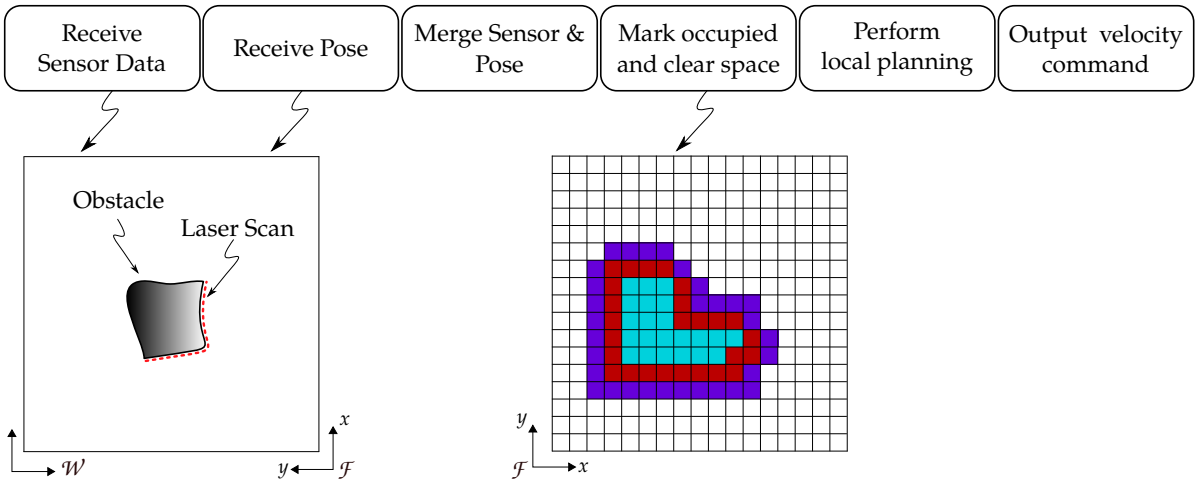


Fig. 2.7 `move_base`'s internal pipeline with emphasis on costmap generation. $\mathscr{W}$ represents the absolute world frame of reference, $\mathscr{F}$ the Frénet frame (robot frame)

If trying to fit the DRHMP algorithm on the above pipeline of Fig. 2.7, it would be placed in the second last box. However, directly using those costmaps for computing obstacle avoidance constraints in the DRHMP is not straightforward. The solution that worked best can be described in two steps:

First, the occupancy grid is inflated according to a linear function that goes from the highest possible cost value at an occupied position in the grid, to zero at a position located at a distance equals to the radius of the robot.

Secondly, bicubic interpolations of the costmap around sample points taken along a trajectory candidate are performed. Those interpolations provide a locally defined, continuous, and differentiable distance function for each sample during optimization. A representation of this approach for one sample is displayed in Fig. 2.8.

The importance of having continuously differentiable distance functions comes from the way the SLSQP solver searches for a solution: it uses finite differences to estimate first and second order derivatives of constraints. Without interpolations, those finite differences

would usually evaluate to zero because the differentiation step size $\hbar$ employed by SLSQP is much smaller than the occupancy grid resolution.

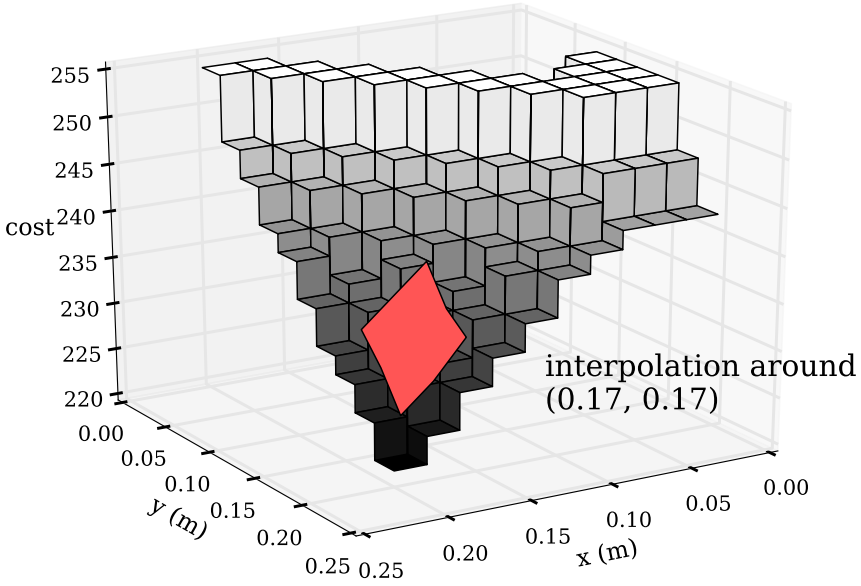


Fig. 2.8 3D representation of the costmap and local interpolation used by the `dhrmp_local_planner`. Cost values of 254 correspond to the detected surface of an obstacle

The costmap interpolation done above can work for simplified robots' footprints (the inflation radius should be representative of the shape of the robot, meaning that aspect ratios far from 1 could be problematic). An advantaged of the method above is that constraints related to obstacle avoidance do not depend on the number of obstacles, only on the number of samples along the trajectory.

However, in our tests done in simulation, that was not the approach used. Because the geometric definition of obstacles can be easily acquired in a simulated environment, we used those representations for evaluating constraints for obstacle avoidance. For minimizing the number of obstacles taken into account in a given moment, we introduced another parameter for the system, which was the sensing reach (d_{sen}) of the robot. At any moment, only the obstacles with a geometric center inside the circle defined by the sensing radius centered at the robot position were used in the optimization. Fig. 2.9 illustrates this parameter.

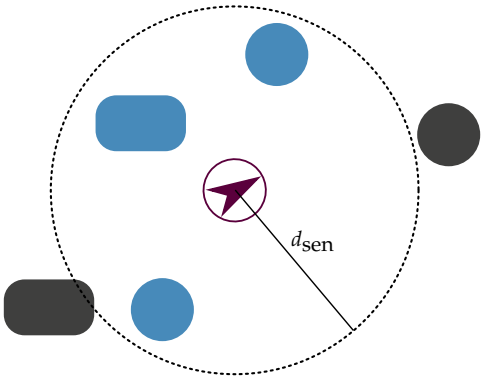


Fig. 2.9 Sensing distance. Only the 3 blue obstacles internal to the circle will be considered as constraints

2.6.5 Definition of Optimization Problems

All throughout this chapter some notation have been used in order to remain generic regarding the optimization problems (NLP, $\text{NLP}_{n,1}$, $\text{NLP}_{n,2}$, $\text{tNLP}_{n,1}$, $\text{tNLP}_{n,2}$). Here below we present the specifics of each formulation for our differential drive robots.

NLP aims to solve the motion from the start to goal configuration in one single optimization. Although it is impractical to use this formulation in reality, as discussed before, it is interesting to see it in details to understand the other optimization formulations. A summary of its definition is shown below first in textual form, second using the mathematical notation from the previous sections. When detailing the optimization problems, we will use strikethrough text to mean constraints or objective functions that are not taken into account by a particular formulation. The objective is to make their differences more apparent.

Objective function:

- Time to reach the goal state (t_f);

Decision variables:

- Control parameters of rotot's trajectory (Γ);
- Time to reach the goal state (t_f).

Constraints:

- Kinematic model (nonholonomic constraints);
- Initial state;

- Goal state (termination constraint);
- Bounded velocities;
- Bounded accelerations;
- Obstacle avoidance;
- Inter-robot collision.

$$
\begin{aligned}
&\underset{(t_f,\Gamma)}{\text{argmin}} && t_f \\
&\text{subject to} && q(0,\Gamma) = q_{\text{start}}, \quad \dot{q}(0,\Gamma) = \dot{q}_{\text{start}}, \\
& && q(t_f,\Gamma) = q_{\text{goal}}, \quad \dot{q}(t_f,\Gamma) = \dot{q}_{\text{goal}}, \\
& && v^2(kT,\Gamma) \leq v^2_{\max}, && k = 0 \dots N-1 \\
& && \omega^2(kT,\Gamma) \leq \omega^2_{\max}, && k = 0 \dots N-1 \\
& && a^2(kT,\Gamma) \leq a^2_{\max}, && k = 0 \dots N-1 \\
& && \alpha^2(kT,\Gamma) \leq \alpha^2_{\max}, && k = 0 \dots N-1 \\
& && \mathrm{d}(O,kT,\Gamma) \geq \epsilon_o, && k = 0 \dots N-1 \\
& && \mathrm{d}(R_d,kT,\Gamma) \geq \epsilon_r, && \forall R_d \in \mathscr{D}, \quad k = 0 \dots N-1
\end{aligned}
$$

Using the trajectory parametrization by B-splines and the flat output property of the system, Γ and t_f become, respectively, the control points and parametric variable of a trajectory in the flat space. v, ω, a and α are derived from Γ and t_f (omitted in some expressions for simplicity) through Eq. (2.21) and (2.22) thus respecting the kinematic constraints of the robot.

Distance functions are represented by d. $\mathrm{d}(O,kT,\Gamma)$ is the minimum distance between the robot and obstacles[5]. $\mathrm{d}(R_d,kT,\Gamma)$ measures the spatio-temporal distance between two robots. R_d represents the information from another robot, namely its intended trajectory and footprint. Robot R_d belongs to the set $\mathscr{D}$, which contains all conflicting robots with respect to the robot computing the trajectory. The criterion used for classifying a robot as belonging to a conflicting set is based on the maximum linear velocity and planning horizon of the robots (see Fig. 2.10). To compute these distances, we evaluate the planned configuration of each robot at each sampled time.

ϵ_o and ϵ_r are minima clearance from obstacles and other robots, respectively. In the experimental section 2.7 that follows, we will discuss these values more. They were kept very

[5] in simulation we compute the distance to all obstacles in the sensing area defined by d_{sen} whereas in the experiments with real robots we use the costmap interpolation

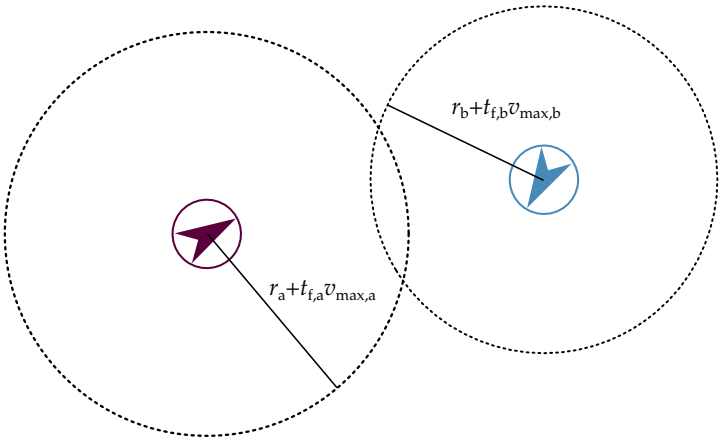


Fig. 2.10 Conflicting robots set. A non empty intersection between the circles means that the robots are mutually present on their conflicting sets $\mathscr{D}$

small or equal to zero during simulations where perfect knowledge of robots and obstacle's positions is possible. In the real experiments, those values where chosen dynamically, during planning, as a way to account for uncertainties in localization and trajectory tracking.

NLP$_{n,1}$ is the problem solved at Step 1 of the DRHMP. Its characteristics, highlighting the ones of the full problem that are not taken into account are :

Objective function:

- ~~Time to reach the goal state (T_f)~~;
- Euclidean distance from planned configuration at $nT_c + T_p$ to the goal configuration.

Decision variables:

- Control parameters of rotot's trajectory (Γ);
- ~~Time to reach the goal state (T_f)~~.

Constraints:

- Kinematic model (nonholonomic constraints);
- Initial state;
- ~~Goal state (termination constraint)~~;
- Bounded velocities;
- Bounded accelerations;
- Obstacle avoidance;

- ~~Inter-robot collision~~.

$$
\begin{aligned}
&\underset{(\Gamma_n)}{\operatorname{argmin}} && \left\| q\left(nT_c + T_p, \Gamma_n\right) - q_{\text{goal}} \right\| \\
&\text{subject to} && q(nT_c, \Gamma_n) = q_{\text{prev}}, \quad \dot{q}(nT_c, \Gamma_n) = \dot{q}_{\text{prev}}, \\
& && v^2(kT + nT_c, \Gamma_n) \leq v_{\max}^2, && k = 0 \ldots N-1 \\
& && \omega^2(kT + nT_c, \Gamma_n) \leq \omega_{\max}^2, && k = 0 \ldots N-1 \\
& && a^2(kT + nT_c, \Gamma_n) \leq a_{\max}^2, && k = 0 \ldots N-1 \\
& && \alpha^2(kT + nT_c, \Gamma_n) \leq \alpha_{\max}^2, && k = 0 \ldots N-1 \\
& && \mathrm{d}(O, kT + nT_c, \Gamma_n) \geq \epsilon_o, && k = 0 \ldots N-1
\end{aligned}
$$

NLP$_{n,2}$ is the problem solved at Step 2 of the DRHMP. Its characteristics are:

Objective function:

- ~~Time to reach the goal state (T_f)~~;
- Euclidean distance from planned configuration at $nT_c + T_p$ to the goal configuration.

Decision variables:

- Control parameters of rotot's trajectory (Γ);
- ~~Time to reach the goal state (T_f)~~.

Constraints:

- Kinematic model (nonholonomic constraints);
- Initial state;
- ~~Goal state (termination constraint)~~;
- Bounded velocities;
- Bounded accelerations;
- Obstacle avoidance;
- Inter-robot collision.

$$
\begin{aligned}
&\underset{(\Gamma_n)}{\text{argmin}} && \left\| q\left(nT_c + T_p, \Gamma_n\right) - q_{\text{goal}} \right\| \\
&\text{subject to} && q(nT_c, \Gamma_n) = q_{\text{prev}}, \quad \dot{q}(nT_c, \Gamma_n) = \dot{q}_{\text{prev}}, \\
& && v^2(kT + nT_c, \Gamma_n) \leq v_{\max}^2, && k = 0 \ldots N-1 \\
& && \omega^2(kT + nT_c, \Gamma_n) \leq \omega_{\max}^2, && k = 0 \ldots N-1 \\
& && a^2(kT + nT_c, \Gamma_n) \leq a_{\max}^2, && k = 0 \ldots N-1 \\
& && \alpha^2(kT + nT_c, \Gamma_n) \leq \alpha_{\max}^2, && k = 0 \ldots N-1 \\
& && \mathrm{d}(O, kT + nT_c, \Gamma_n) \geq \epsilon_o, && k = 0 \ldots N-1 \\
& && \mathrm{d}\left(R_d, kT + nT_c, \Gamma_n\right) \geq \epsilon_r, && \forall R_d \in \mathscr{D}, \quad k = 0 \ldots N-1
\end{aligned}
$$

In the DRHMP approach the conflicting robots set $\mathscr{D}$ is defined at each iteration based on the circles with radius $r + T_p v_{\max}$ centered at $q(nT_c, \Gamma_{n-1})$.

tNLP$_{n,1}$ is the optimization problem solved in the Step 1 of the termination stage.

Objective function:

- Time to reach the goal state (T_f);
- ~~Euclidean distance from planned configuration at $nT_c + T_p$ to the goal configuration~~.

Decision variables:

- Control parameters of rotot's trajectory (Γ);
- Time to reach the goal state (T_f).

Constraints:

- Kinematic model (nonholonomic constraints);
- Initial state;
- Goal state (termination constraint);
- Bounded velocities;
- Bounded accelerations;
- Obstacle avoidance;
- ~~Inter-robot collision~~.

$$
\begin{aligned}
&\underset{(T_f,\Gamma_n)}{\text{argmin}} && T_f \\
&\text{subject to} && q(nT_c,\Gamma) = q_{\text{start}}, \quad \dot{q}(nT_c,\Gamma) = \dot{q}_{\text{start}}, \\
& && q(nT_c+T_f,\Gamma) = q_{\text{goal}}, \quad \dot{q}(nT_c+T_f,\Gamma) = \dot{q}_{\text{goal}}, \\
& && v^2(kT+nT_c,\Gamma) \le v^2_{\max}, && k=0\ldots N-1 \\
& && \omega^2(kT+nT_c,\Gamma) \le \omega^2_{\max}, && k=0\ldots N-1 \\
& && a^2(kT+nT_c,\Gamma) \le a^2_{\max}, && k=0\ldots N-1 \\
& && \alpha^2(kT+nT_c,\Gamma) \le \alpha^2_{\max}, && k=0\ldots N-1 \\
& && \mathrm{d}(O,kT+nT_c,\Gamma) \ge \epsilon_o, && k=0\ldots N-1
\end{aligned}
$$

tNLP$_{n,2}$ is the optimization problem solved in the Step 2 of the termination stage.

Objective function:

- Time to reach the goal state (T_f);
- ~~Euclidean distance from planned configuration at $nT_c + T_p$ to the goal configuration~~.

Decision variables:

- Control parameters of rotot's trajectory (Γ);
- Time to reach the goal state (T_f).

Constraints:

- Kinematic model (nonholonomic constraints);
- Initial state;
- Goal state (termination constraint);
- Bounded velocities;
- Bounded accelerations;
- Obstacle avoidance;
- Inter-robot collision.

$$
\begin{aligned}
\underset{(T_f,\Gamma_n)}{\text{argmin}} \quad & T_f \\
\text{subject to} \quad & q(nT_c,\Gamma) = q_{\text{start}}, \quad \dot{q}(nT_c,\Gamma) = \dot{q}_{\text{start}}, \\
& q(nT_c+T_f,\Gamma) = q_{\text{goal}}, \quad \dot{q}(nT_c+T_f,\Gamma) = \dot{q}_{\text{goal}}, \\
& v^2(kT+nT_c,\Gamma) \le v^2_{\max}, && k = 0\dots N-1 \\
& \omega^2(kT+nT_c,\Gamma) \le \omega^2_{\max}, && k = 0\dots N-1 \\
& a^2(kT+nT_c,\Gamma) \le a^2_{\max}, && k = 0\dots N-1 \\
& \alpha^2(kT+nT_c,\Gamma) \le \alpha^2_{\max}, && k = 0\dots N-1 \\
& \mathrm{d}(O,kT+nT_c,\Gamma) \ge \epsilon_o, && k = 0\dots N-1 \\
& \mathrm{d}\left(R_d,kT+nT_c,\Gamma\right) \ge \epsilon_r, && \forall R_d \in \mathscr{D}, \quad k = 0\dots N-1
\end{aligned}
$$

In order to simplify the the following sections notation, we may consider $\tau_k \equiv kT + nT_c$.

2.6.6 Multi-robot Communication

The wireless communication among robots is, in itself, a very challenging topic. Whatever the network architecture and employed technology may be, the robots need to be able to exchange information with minimum latency possible at least once every T_c interval. Furthermore, the robots' clocks need to be very well synchronized so that timestamps are consistent among the whole multi-robot system. Another practical concern is about how the middleware ROS used for the experiments handles communication among nodes distributed across different machines over a network[6].

For the simulated tests, we used distributed shared memory among different threads to exchange information with negligible communication delays. For experiments with real robots, different setups were tested. Concerning the network architecture, a star topology was first tested using a standard Wifi router. Due to high latencies, we switched to an AdHoc network where the two robots could communicate directly.

Regarding the middleware, the use of one single ROS Master instance was quickly showed to be unpractical, and a custom communication bridge was created. Finally, for clock synchronization, we used chrony utility [18].

[6]This concern may not apply to the most recent version of ROS, ROS 2 that is built on top of DDS (Data Distribution Service)

2.6.7 Localization and Tracking Error

Localization of vehicles used in the experiments was done by using a particle filter [83] merging data from gyroscopes, encoders, rangefinder sensors, and static maps. As is for any localization system, there is an inherent error associated with the robot's estimated configuration. That uncertainty can be characterized by the covariance matrix of the robot's configuration, which is known for each robot at each instant τ_k. Likewise, each robot can estimate its tracking error based on the planned reference trajectory and its configuration estimate.

Both covariance values and tracking errors were sent as part of the information exchanged between robots, along with their estimated configuration and planned trajectory. This exchange enables a given robot R to compute a conservative "safety distance" from robot R_c that is time-dependent and can replace the constant $\epsilon_r(R_c)$ in the NLP constraints.

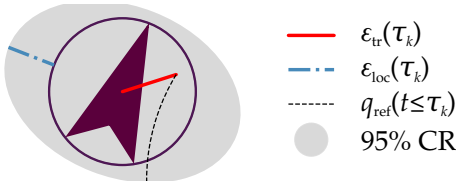


Fig. 2.11 Localization and tracking errors representation. The 95% Confidence Region ellipse encompasses the central 95% of the probability mass of possible locations for a robot at instant τ_k. The ellipse is dilated by the robot's radius.

The new value is computed according to the equation:

$$\epsilon_r(R_c, t) = \epsilon_{\text{tr}}(t) + \epsilon_{\text{loc}}(t) + \epsilon_{\text{tr},R_c}(t) + \epsilon_{\text{loc},R_c}(t) \tag{2.25}$$

where ϵ_{tr} is simply the euclidean norm of lateral and longitudinal tracking errors and ϵ_{loc} is a confidence value related to position estimates. ϵ_{loc} depends on the covariance matrix given by the robots' localization modules and its expression is given by Eq. (2.26):

$$\epsilon_{\text{loc}}(t) = \sqrt{\chi^2_\nu(2)\|\lambda_0, \lambda_1\|_\infty} \tag{2.26}$$

with λ_0 and λ_1 being the eigenvalues of the covariance matrix associated with the bivariate normal distribution of possible locations in the XY plane of a robot at an instant τ_k. $\chi^2_\nu(2)$ is the 2 degrees of freedom chi-square value for a $(1-\nu)$ confidence region (CR).[7]

[7] A 95% CR (i.e. $\chi^2_{0.05}(2) = 5.991$) was used in the experiments

Similarly, ϵ_o in the NLP constraints can be replaced by $\epsilon_o(t)$, where:

$$\epsilon_o(t) = \epsilon_{\text{tr}}(t) + \epsilon_{\text{loc}}(t) \tag{2.27}$$

Fig. 2.11 shows a graphical representation of errors $\epsilon_{\text{tr}}(t)$ and $\epsilon_{\text{loc}}(t)$ at instant τ_k.

2.7 Experimental Evaluation

We will now present experimental results of this framework applied to a simple multi-robot simulation first, then to two real robots.

For performing DRHMP, a reasonable number of parameters have to be set. These parameters can be categorized into two groups. **Algorithm related** parameters and the **optimization solver related** ones. Among the former group, the most important ones are:

- The number of sample for time discretization (N);
- The number of internal knots for the B-spline curves (N_{knots});
- The planning horizon for the sliding window (T_p);
- The computation horizon (T_c);
- The detection radius of the robot (d_{sen}).

The latter kind depends on the numeric optimization solver adopted. However, since most of them are iterative methods, it is common to have at least a maximum number of iterations and a stop condition parameters. This large number of parameters makes the search for a satisfactory set of parameters' values a laborious task. Therefore, it is crucial to have a better understanding of how the changes in these parameters impact performance criteria as we will show in the remainder of this chapter.

2.7.1 Kinematic Simulation

We first applied our approach to a simple kinematic simulation that simulates multiple robot motion among a map of polygonal obstacles in order to demonstrate the multi-robot collision-free planning ability. Since it is a simple kinematic simulation, ϵ_r is considered constant and equals to zero.

2.7.1.1 Resulting Trajectories

We first illustrate the performance of our algorithm qualitatively, using optimized parameters whose influence will be discussed in the next section. The trajectories and velocities shown

in Fig. 2.12 and 2.13 illustrate the motion planning solutions found by the two steps of our approach for a team of three robots. Their motion is planned in an environment where three static obstacles are present. Each point along the trajectory line of a robot represents the beginning of a T_c update/computation horizon.

It is possible to see on those figures how the planner generates configuration and input trajectories satisfying the constraints associated with the goal states that are reached by the three robots.

Figure 2.12 presents the resulting plan computed ignoring coupling constraints (Step 2 is not performed), and consequently, two points of collision occur. After the application of the second step, including constraints to avoid robot collisions, a collision-free solution is found, presented in Figure 2.13. Especially near the regions were collisions occurred, a change in the trajectory is visible between Figure 2.12 to Figure 2.13 to avoid collision. Additionally, changes in the robots (linear) velocities across charts in both figures can be noticed. Finally, the bottom charts show that the collisions were indeed avoided: inter-robot distances in Figure 2.13 are greater than or equal to zero all along the simulation.

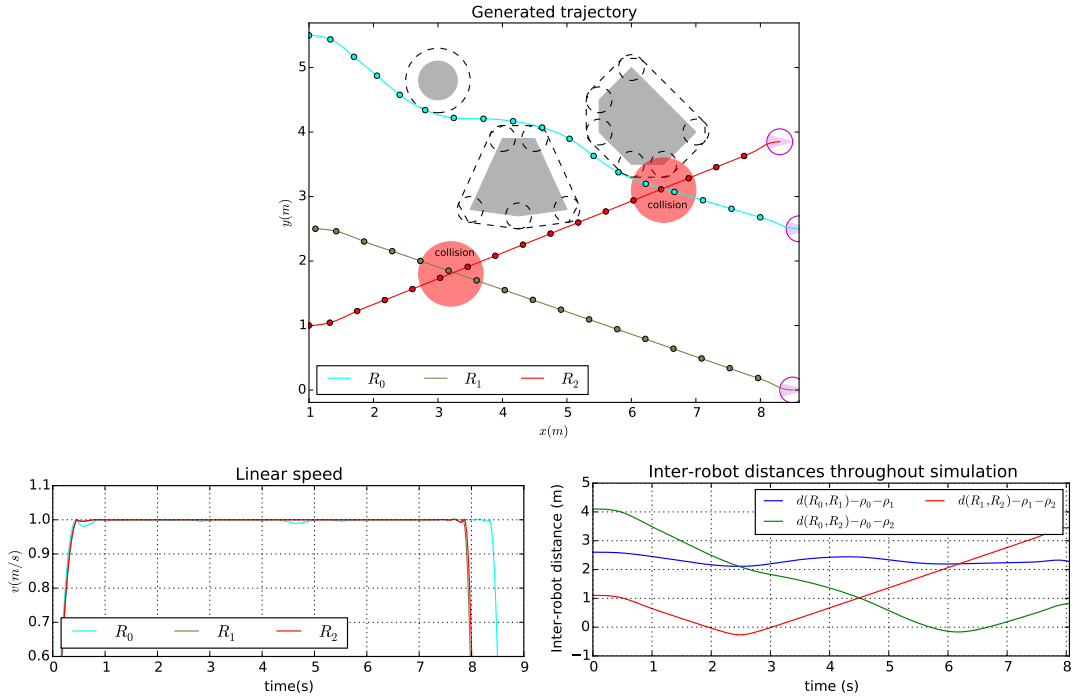


Fig. 2.12 Motion planning solution without collision handling

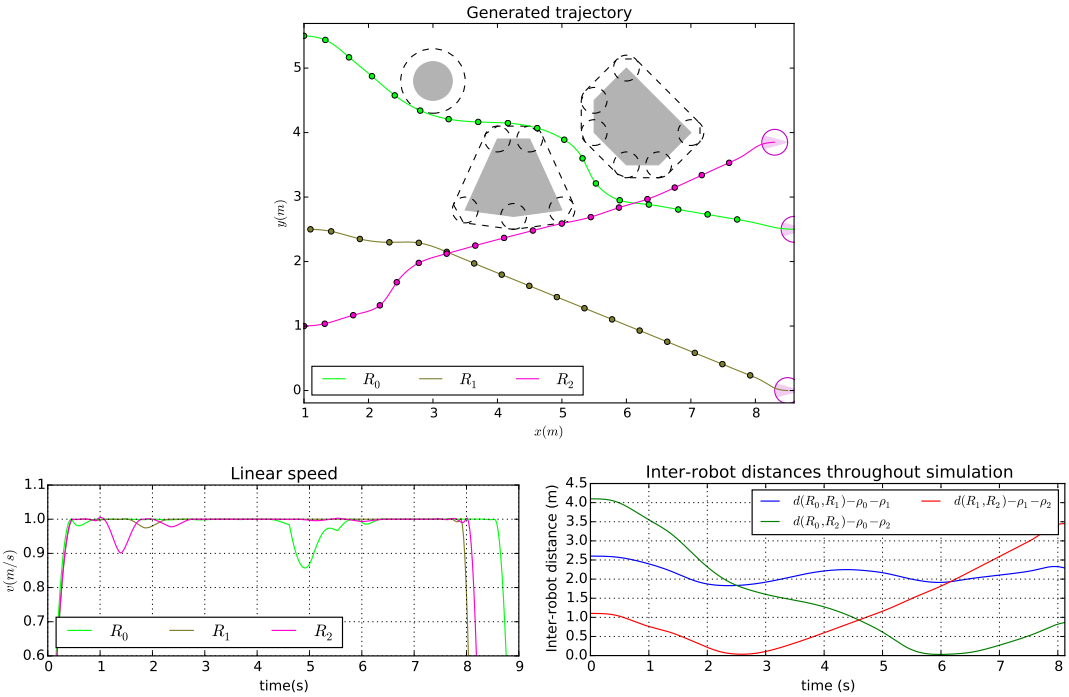


Fig. 2.13 Motion planning solution with collision handling

2.7.1.2 Parameters' Impact

We will now study how the **algorithm related** parameters and the **optimization solver related** ones influence the approach performance.

Three criteria considered important for the validation of this method were studied:

- Maximum computation time during the planning over the computation horizon (MCT/T_c ratio);
- Obstacle penetration area (P);
- Travel time (t_f).

Different parameters configuration and scenarios where tested in order to highlight how they influence those criteria.

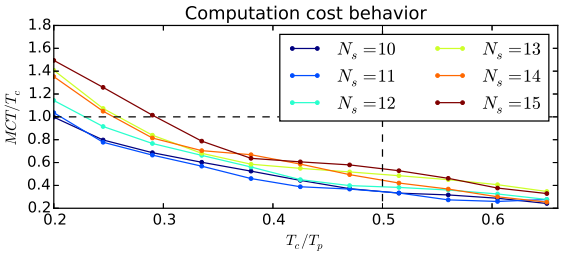
Maximum computation time over the computation horizon

The significance of this criterion lies in the need for assuring the real-time property of this algorithm. In a real implementation of this approach, the computation horizon would always have to be superior to the maximum time needed for computing a plan (MCT < T_c).

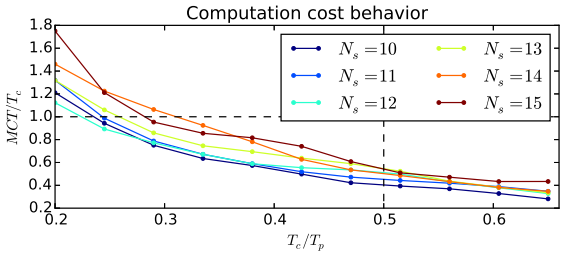
Table 2.1 Values for scenario definition

$v_{\max}$	1.00 m/s
$\omega_{\max}$	5.00 rad/s
q_{start}	$[-0.05\ 0.00\ \pi/2]^T$
q_{goal}	$[0.10\ 7.00\ \pi/2]^T$
u_{start}	$[0.00\ 0.00]^T$
u_{goal}	$[0.00\ 0.00]^T$
O_0	[0.55 1.91 0.31]
O_1	[−0.08 3.65 0.32]
O_2	[0.38 4.65 0.16]

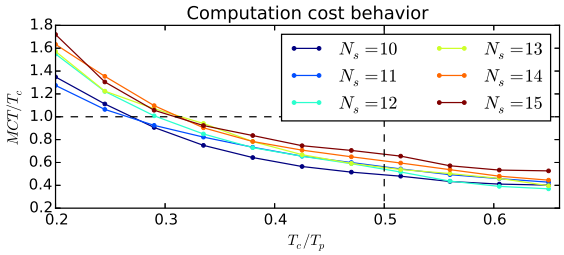
Table 2.1 summarizes the parameters of the scenario studied for a single robot to analyse this criteria.



(a) Four internal knots



(b) Five internal knots



(c) Six internal knots

Fig. 2.14 Maximum computation time over computation time horizon in a three obstacles scenario simulations

Results obtained from simulations in that scenario are presented in Figure 2.14, for different parameters set. Each dot along the curves corresponds to the average of MCT/T_c along different T_p's for a given value of (T_c/T_p, N). The absolute values observed in the

charts depend on the processing speed of the machine where the algorithm is run. Those simulations were run on an Intel Xeon CPU 2.53GHz processor.

Rather than observing the absolute values, it is interesting to analyze the impact of changes in the parameters values. In particular, an increasing number of time samples N increases MCT/T_c for a given T_c/T_p. Similarly, an increasing of MCT/T_c as the number of internal knots N_{knots} increases from charts 2.14a to 2.14c is noticed.

Further analyses of those data show that finding the solution using the SLSPQ method requires $O(N_{\text{knots}}^3)$ and $O(N)$ time. Although augmenting N_{knots} can yield to impractical computation times, typical N_{knots} values did not need to exceed 10 in our simulations, which is a sufficiently small value.

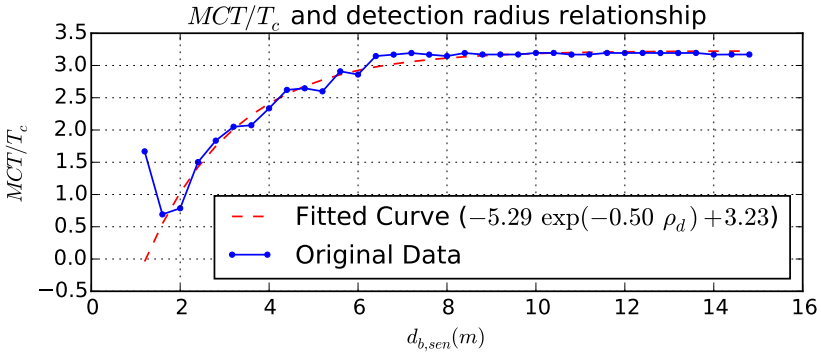


Fig. 2.15 Increasing of detection radius and impact on a MCT/T_c ratio

Another parameter having direct impact on the MCT/T_c ratio is the detection radius of the robot's sensors. As the detection radius of the robot increases, more obstacles are seen at once which, in turn, increases the number of constraints in the optimization problems. The impact of increasing the detection radius d_{sen} in the MCT/T_c ratio can be seen in Figure 2.15 for a scenario with seven obstacles. The computation time stops increasing as soon as the robot sees all obstacles present in the environment.

Obstacle penetration

Obstacle penetration area P gives a metric for obstacle avoidance and, consequently, for the solution quality. A solution where the planned trajectory does not pass through an object at any instant of time gives $P = 0$. The solution quality decrease with increasing P. However, since time sampling is performed during the optimization, P is usually greater than zero (see Fig.2.16). A way of assuring $P = 0$ would be to increase the radius of the obstacle computed by the robot's perception system by the maximum distance that the robot can run within the time spam T_p/N. However simple, this approach represents a loss of optimality and is not considered in this work.

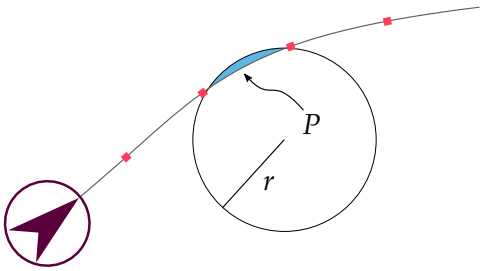


Fig. 2.16 Penetration area illustration. r represents the obstacle plus the robot radii, the red dots along the trajectory represent the sampled points for optimization

It is relevant then to observe the impact of the algorithm parameters in the obstacle penetration area. T_c/T_p ratio, N_{knots} and d_{sen} impact on this criteria is only significant for degraded cases, meaning that around typical values those parameters do not change P significantly. However, time sampling N is a relevant parameter. Figure 2.17 shows the penetration area decreasing as the number of samples increases.

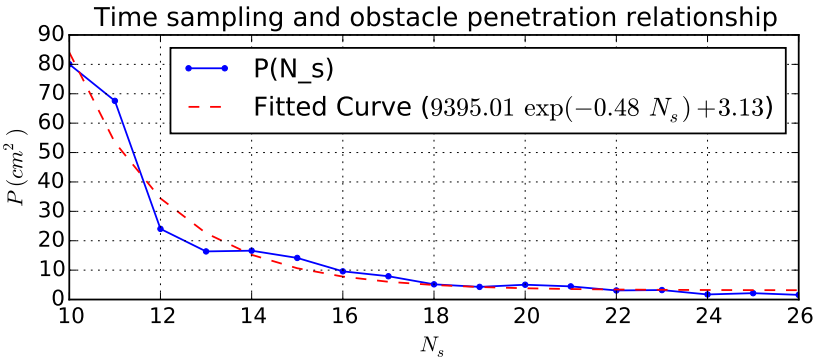


Fig. 2.17 Obstacle penetration decreasing as sampling increases

Travel time t_f

Another complementary metric for characterizing solution quality is the travel time t_f. Analyses of data from several simulations show a tendency that for given values of N_{knots}, N and T_c the travel time decreases as the planning horizon T_p decreases. This can be explained by the fact that a higher sampling density yields more optimal solutions (in terms of travel time).

Another relevant observation is that the overall travel time is shorter for smaller N's. This misleading improvement comes from the fact that the fewer the samples the greater will be the obstacle penetration area as shown previously in Figure 2.17, and therefore the shorter the computed path.

Furthermore, Figure 2.18 shows travel time invariance for changes in the detection radius far from degraded values that are too small. This fact points out that local knowledge of the environment provides enough information for finding good solutions.

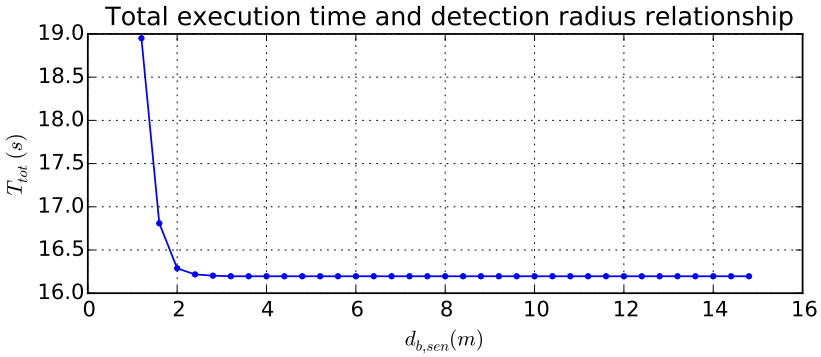


Fig. 2.18 Increasing of detection radius and impact on t_f

2.7.2 Real Robots

For the rest of this section, we present the results of experiments performed on two real mobile robots (two Turtlebots 2).

2.7.2.1 Turtlebot 2 Mobile Platform

Turtlebot was designed in 2011 as a minimalist platform for ROS-based mobile robotics education and prototyping. It has a small differential-drive mobile base (Fig.2.19) with an internal battery, power regulators, and charging contacts. Atop this base is a stack of laser-cut “shelves” that provide space to hold a netbook computer, depth camera, and lots of open space for prototyping. To control cost, Turtlebot relies on a depth camera for range sensing; it does not have a laser scanner. Despite this, mapping and navigation can work quite well for indoor spaces. Turtlebots are available from several manufacturers for less than $2,000[8].

The Turtlebots were equiped with an Asus Xtion Pro Live 3D Sensor (RGBD camera, Fig. 2.20). The field of view of the camera is 58° horizontal, 45° vertical.

2.7.2.2 Experimental Setup

Experiments were carried out in order to investigate two aspects: how the DRHMP compares to another local MP in a "single robot avoiding an obstacle" situation and how well

[8]More information is available at http://turtlebot.com

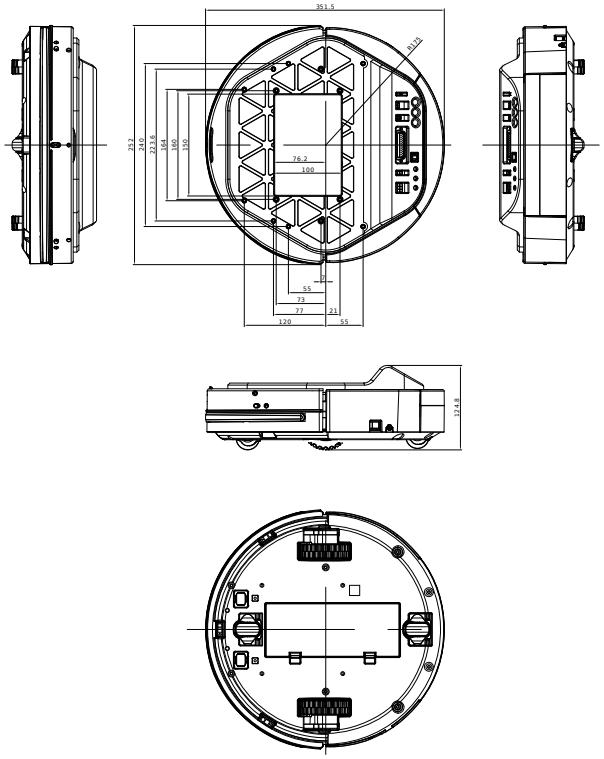


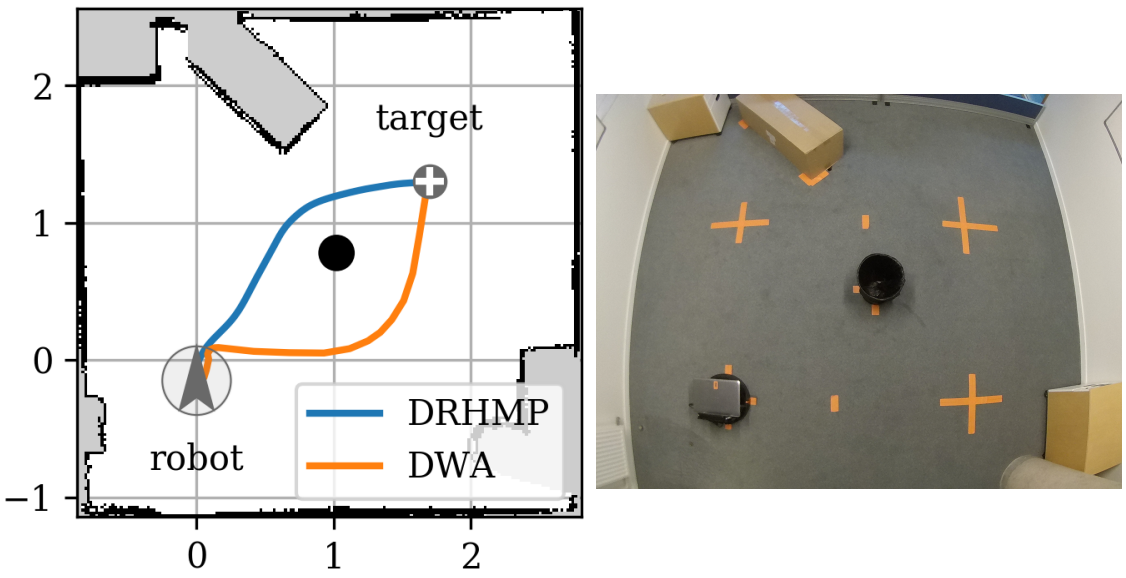
Fig. 2.19 Kuboki mobile base used in experiments

Fig. 2.20 Asus Xtion Pro Live 3D sensor

collision avoidance between two robots running the DRHMP is performed in face of real communication, perception and trajectory tracking issues. Across all experiments we used $v_{\max} = 0.2\ m/s$, $a_{\max} = 0.5\ m/s^2$, $T_p = 3.8\ s$, $T_c = 0.3\ s$. A simple controller designed for tracking a reference admissible trajectory based on the kinematic model of the system was used [77].

2.7.2.3 Experiment 1: Single Robot Obstacle Avoidance

A testbed as shown in Fig. 2.21 was used for comparing collision avoidance with a static obstacle using the DRHMP approach and the well known Dynamic Window approach (DWA) [26] natively implemented in ROS. Although admittedly, each planners' performance can be highly impacted by the configuration of its parameters, an effort was made to set those values so equivalent behaviors could be obtained. Velocity and acceleration limitations were set to the same values for instance.



(a) (b)

Fig. 2.21 Experiment 1. (a) shows a representation of the testbed using the same map information as the localization system of the real robots. (b) is an actual photo of the testbed

As indicated in Fig. 2.21, the robot located near the XY origin had to reach the "target" point located about 2.3 m of its initial position. An obstacle of about 20 cm in diameter not known by the robot in advance was placed in the way (the obstacle is not visible to the robot from the starting position because of the limited camera field of view). As the robot

senses its environment through his depth camera, the obstacle and the room are taken into account.

Table 2.2 Experiment 1 summary

	DRHMP		DWA	
	Mean	Std	Mean	Std
travel time (s)	12.637	0.084	18.790	0.385
average linear speed (m/s)	0.193	0.000	0.136	0.002
final position error (m)	0.014	0.004	0.092	0.013
final yaw error (rad)	0.011	0.004	0.355	0.053
min clearance from obstacles (m)	0.069	0.014	0.228	0.023

Table 2.2 summarizes the performance of each planner regarding five criteria. Mean and standard deviation (Std) data were based on 10 trials for each algorithm. In none of the tests, neither MP approaches failed to avoid obstacles. Compared to DWA, DRHMP can be seen as a less conservative approach. It keeps less clearance from obstacles and can produce a higher average linear velocity in order to minimize travel time. That behavior derives from the type of objective function used in the NLPs.

Similarly, DWA's behavior derives from its scoring algorithm. It is worth noticing that DRHMP presents an inferior standard deviation compared to DWA, which suggests it is a more stable approach. Typical paths adopted by both algorithms can be seen in Fig. 2.21a.

2.7.2.4 Experiment 2: Multi-robot Motion Planning

The second experiment consisted of having two robots going successively from one target location in their shared workspace to another. Those targets positions were such that the robots would execute two different triangle-shaped loops that share a common side. Along this shared side, the two robots (if the timing was right) would have to cross each other to reach their next target. The robots' trajectories in Fig. 2.22 illustrate this setup.

To better evaluate the DRHMP performance in that scenario, two different sub-experiments were set. In the first one (2a), a simplification about tracking and localization errors was made. They were considered equal to zero by the DRHMP algorithm running in both robots ($\epsilon_r = 0$). It implied though that the physical robots had to be kept still during planning to prevent them from colliding. As we will see, the $\epsilon_r = 0$ assumption is far from realistic, provided the platforms we worked with. The second sub-experiment (2b), on the contrary, makes no such simplification and ϵ_r is based on the real information about the physical robots executing the planned trajectories. This second case shows how even with errors of about 50 cm the DRHMP can safely find collision-free trajectories for both physical

robots. Everything else, especially communication, is done in the same way across both experiments, as described in subsection 2.6.6.

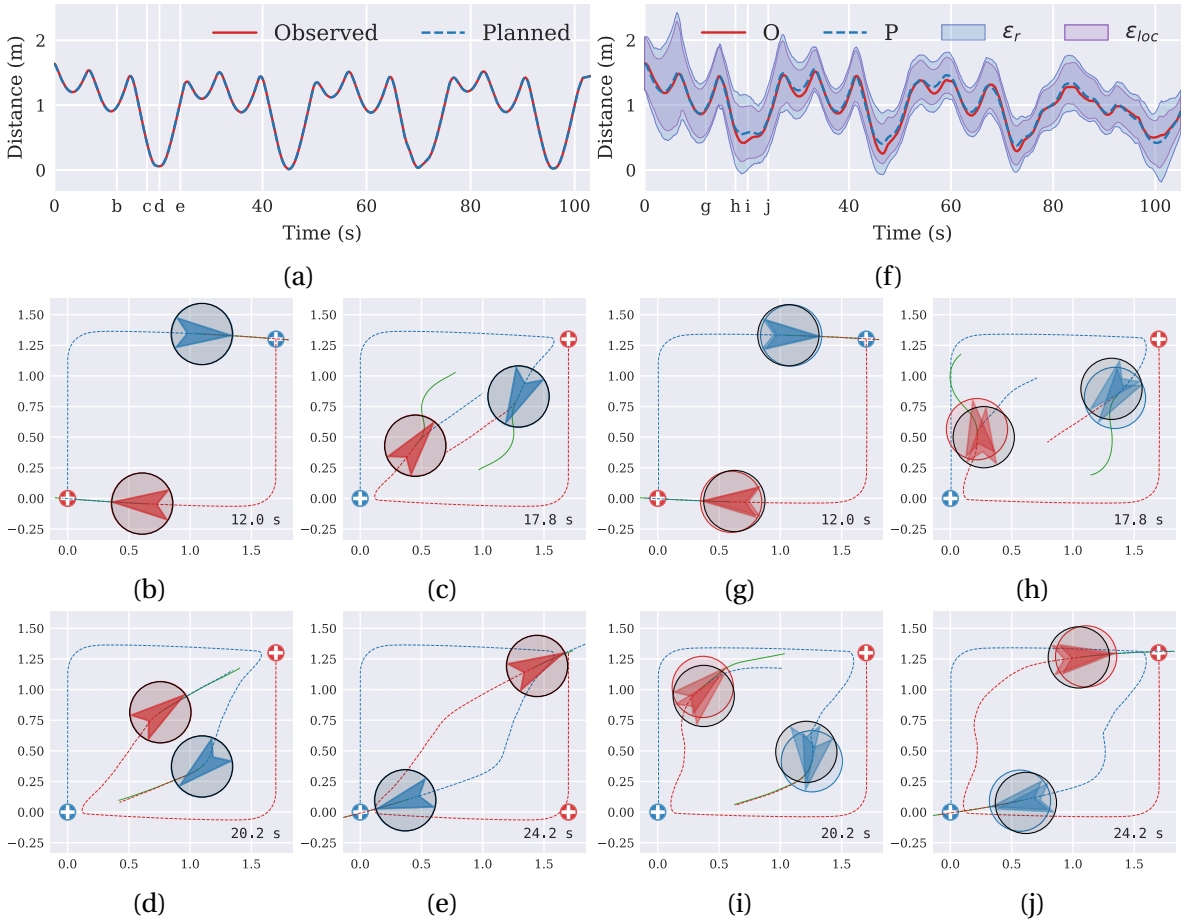


Fig. 2.22 Experiment 2 results. (a) to (e) refer to sub-experiment 2a while (f) to (j) to sub-experiment 2b. "Planned" or "P" data shows inter-root distance computed based on the planned trajectories generated by the DRHMP approach. "Observed" or "O" data represents the same distance but based on observed/estimated actual position of the robots. In (f) the blue and purple error bands centered around the "Planned" line represent respectively the safety distance of Eq. (2.25) (ϵ_r) and its localization component ($\epsilon_{\text{loc},R_a} + \epsilon_{\text{loc},R_b}$).

Experiment without localization error

The trajectories produced and the inter-robot distance along this entire sub-experiment can be seen in Figs. 2.22a to 2.22e. Planned trajectories would allow both robots to avoid each other with almost no clearance at four different moments, as indicated by the inter-robot distance curve passing near zero in Fig. 2.22a.

Figs. 2.22b to 2.22e represent four snapshots of both robots planning processes around the first moment of collision avoidance. The continuous green lines in front of the robots

represent the XY points of their planned trajectory for the horizon T_p (i.e. step2-generated trajectories). The dashed line in front of a given robot R_a represents what the other robot R_b thinks R_a will do (i.e. R_a's step1-generated trajectory as known by R_b as a result of their communication).

When collision is not an issue, dashed lines may superpose almost exactly the green continuous ones[9]. In contrast, as collision becomes an issue, one can observe a greater difference between those two trajectories. At Fig. 2.22c, R_a's step1-generated trajectory as known by R_b is shown as coming straight at R_b. R_a's step2-generated trajectory, in green, is quite different from that and clearly avoids a future collision. That is because R_a has already taken into account R_b's step1-generated trajectory into its Step 2.

Following the time sequence and observing Fig. 2.22d, one can see that the actually followed paths after solving the conflict reflect a smaller deviation from the otherwise straight-line trajectory when compared to those planned green lines in Fig. 2.22c. This is due to the Receding Horizon nature of the approach, which interleaves planning and execution and therefore reconsiders the avoidance trajectory adapting to the changes made by the other robot. Its implication, albeit conditioned to T_c values, is that collision avoidance ends up being achieved with near the minimum clearance possible (zero in this case). Furthermore, a natural compromise between robots is achieved: they both deform their initial straight trajectories of comparable amounts.

That is precisely the behavior expected for the DRHMP in a multi-robot scenario and observed in the simulation previously presented.

Experiment with real localization

In this sub-experiment, the physical robots did actually navigate their workspace and the real observed information about their localization and trajectory tracking was used in the DRHMP.

Due to imprecisions of sensors, actuators and low level controllers, ϵ_r becomes meaningful in this second case, as shown by the error bands in Fig. 2.22f. They have the effect of reducing the space of acceptable solutions in which the DRHMP searches for optimal trajectories. If additionally, the workspace is very cluttered, there may be no acceptable solutions left yielding optimization errors at the SLSQP algorithm level.

Nevertheless, for the studied setup, acceptable solutions were always found and the inter-robot collisions of the physical robots were prevented at all times throughout the experiment (as observed in the experiment video in [55]).

[9] communication delays may still prevent them from being identical

From Fig. 2.22g to 2.22j, the difference between black and colored circles representing the robots reflects the tracking error. Black circles use the mean information of robots' localization systems while colored ones use the planned poses. As before, dashed and continuous green lines in front of a given robot represent respectively the step1-generated trajectory known by the other robot and its own step2-generated trajectory.

The utility of a nonconstant ϵ_r taking robots localization uncertainty into account can be appreciated in Fig. 2.22f. It can be observed that when ϵ_r takes smaller values (such as near time 45 s compared to time 20 s), the inter-robot distance can be reduced as well.

Table 2.3 Experiment 2 summary

			Min	Max	Mean	Std	Obs
2a	Planned IRD		0.014	-	-	-	-
	R_a	COD	0.000	0.403	0.016	0.044	370
		USD	0.282	1.187	0.472	0.158	370
		URI	64.506	92.586	86.920	5.249	370
	R_b	COD	0.000	0.265	0.017	0.038	323
		USD	0.280	0.895	0.512	0.134	323
		URI	64.204	92.623	85.962	4.714	323
2b	Planned IRD		0.376	-	-	-	-
	Observed IRD		0.253	-	-	-	-
	ϵ_r (m)		0.292	0.933	0.435	0.106	351
	R_a	COD	0.004	0.232	0.014	0.026	355
		USD	0.073	0.973	0.408	0.103	355
		URI	61.112	98.089	88.865	3.608	355
	R_b	COD	0.000	0.186	0.018	0.036	351
		USD	0.201	0.821	0.526	0.064	351
		URI	68.930	94.701	85.660	3.223	351

URI = % of the Received Information that is actually used by the DRHMP at Step 2; **COD** = communication delay measured by the receiving end in seconds; **USD** = delay between receiving and start using RI in DRHMP in seconds; **IRD** = inter-robot distance in meters

Table 2.3 summarizes statistics of experiment 2 regarding communication and inter-robot distances. Comparing communication-related values between robots R_a and R_b and then between sub-experiments 2a and 2b shows indeed that communication conditions were very symmetric. The column titled "Obs" shows the number of observations and it is roughly equal to the duration of the experiment divided by T_c (the period the robots exchange step1-generate trajectories). Furthermore, due to the asynchronism between robots' planning processes and communication delays, it is common that part of the trajectory information received by a robot concerns a time interval of no interest to that

robot. In other words, at Step 2 of the DRHMP, the information about another robot's trajectory may be partially too old or planned for too far into the future. The percentage of the information that can actually be used is referred to as URI.

Overall, despite considerable communication delays (tens of ms), low-quality localization (main responsible for high ϵ_r values) and use of a simple kinematic controller that does not take sliding and actuator response times into account, the DRHMP manages to produce satisfying results with respect to multi-robot collision avoidance.

2.8 Conclusions

In this chapter, we presented a Distributed Receding Horizon Motion Planning (DRHMP) approach for planning multiple robot motions in dynamic environments. At the goal configuration neighborhood, the receding horizon approach ceases and a termination planning problem is solved for bringing the robots to their precise final state.

Key techniques for implementing this motion planning approach exploited the system flatness property, B-spline parameterization of trajectories, use of SLSQP optimizer, interpolation of the discrete world representation for obtaining differentiable constraints equations for obstacle avoidance, and online estimation of localization and tracking errors.

Experiments in simulation and on real robots show that the approach is able to work in real-time, efficiently plan collision-free paths, and can deal with real-life communication and localization difficulties.

A missing feature for more real applications would be the implementation of fail-safe measures when no solution trajectory can be found by Step 1 or Step 2. The assumption that the footprint of the vehicle can be described as a circle is a strong one as well. Further work should try to extend the approach to more generic shapes.

This chapter relied on simple control algorithms that will be limited for highly dynamic robots. The next chapter will introduce a new control law that exploits the computed paths in order to improve trajectory tracking performance.

Chapter 3

Trajectory Tracking

From the previous chapter, we have shown how a motion planner can be devised to obtain feasible reference trajectories for a set of vehicles by an association of Mathematical Programming and Receding Horizon Approach in a method called DRHMP.

This chapter addresses the problem of tracking position and orientation given by those reference trajectories for a system that undergoes perturbations. As pointed out in [77] the term *tracking problem* is a rather loose one. Here the tracking problem is understood as it usually is in the context of control literature. It is associated with the problem of asymptotically stabilizing the system state around the reference trajectory. The diagram depicted in Fig. 3.1 shows one possible architecture where the addition of a controller could be used to that end.

So, we will investigate two ways to optimally follow the admissible reference trajectory. The first kinematic approach that is put forward globally asymptotically stabilizes the system around the trajectory under the condition the orientation error between the physical robot and the reference trajectory be smaller than $\pi/2$. The second approach is a non-linear predictive control for path tracking that takes into account lateral wheel slippage.

For these two controllers, we consider a sufficiently simple two-dimensional model in the yaw plane.

3.1 Problem Overview

We remind here that at the discretized time kT_c, k being a strictly positive integer, the DRHMP algorithm outputs a reference trajectory for a time horizon $[kT_c, kT_c+T_p]$, with $T_c \ll T_p$. On this time horizon, only the part of the trajectory corresponding to the computation horizon T_c is kept as a section of the final reference trajectory, the rest $(T_p - T_c)$ being

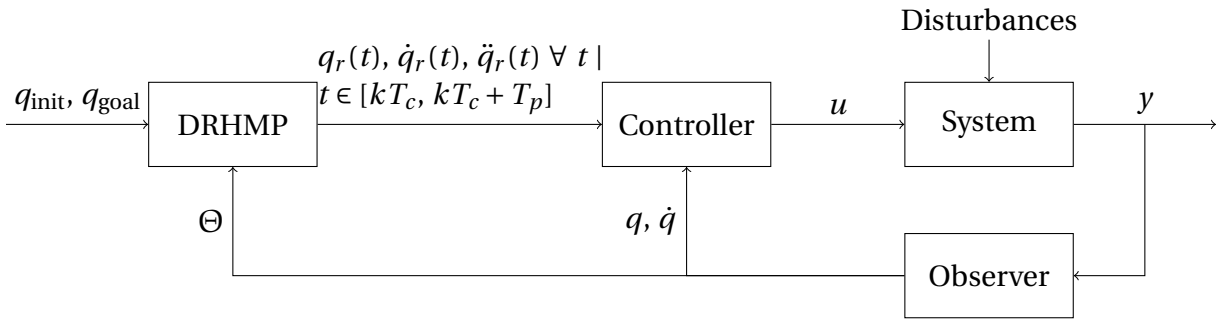


Fig. 3.1 The planner, knowing the initial and goal configurations of the robot, takes information about obstacles and other robots (denoted by Θ) from the observer and outputs a reference trajectory. The controller takes a state feedback from the observer and the reference trajectory from the planner and generates the system's input u.

replaced by a new trajectory when the horizon $[(k+1)T_c, (k+1)T_c + T_p]$ is computed (see Fig. 2.10).

The generated trajectory is a cubic B-spline of degree 3, which is of class C^2, provided that the connection nodes are all distinct. Thus, the expression of this trajectory $q_r(t)$ and its two derivatives $\dot{q}_r(t)$ and $\ddot{q}_r(t)$ are available at any instant inside the time window $[kT_c, kT_c + T_p]$.

The problem is thus to define a control law to track this reference trajectory, which is gradually defined over a sliding time horizon. We can, therefore, consider this to be a classic problem of pursuing non-stationary trajectories. In the case of the plane, this consists of determining a control to asymptotically stabilize the longitudinal, lateral, and directional errors of the robot $(e_x(t), e_y(t), e_\psi(t))$.

In his work [20], Defoort proposes a second-order sliding-mode control law with integral action, in order to solve the problem of practical trajectory stabilization, with practical stabilization less strong than asymptotic stabilization corresponding to stabilization within a restricted and known domain [41]. One difficulty in implementing this method is its setting, which requires knowing the higher-order derivatives of the slip variable. Another more general limitation related to the theory of sliding mode controllers is the chattering phenomenon that causes high-frequency oscillations, increasing energy consumption, and damage to the actuators. To limit the chattering in control signals, a second-order sliding mode controller may be investigated, as already done in [30], for instance.

We propose here to investigate a different approach, by testing two types of controllers, with complementary benefits and objectives, described in detail in the following sections.

3.2 Tracking a Reference Vehicle with same Kinematics

At first, let us investigate the pursuit of the reference trajectory $q_r(t)$ using a control law based on the kinematic model of robots. This Tracking Reference Vehicle with Same Kinematics (TRVSK) strategy, which is often sufficient, has the major advantage of being easy to implement. It thus allows a first evaluation of the motion planning strategy developed in the previous chapter.

Following we present the kinematic control law developed by Thuilot [84], based on the assumption of rolling without slipping, established from Samson's transformation into a chained system [75].

This is an adaptive backstepping controller as proposed in various previous works [4, 8, 42, 50, 51, 69]. Adaptive controllers have the specificity to improve their performance by estimating unknown or varying parameters.

The one used here is model-based. It has the advantage of being designed based on a chained form model. That way, linear systems theory is still applicable, while still considering the actual nonlinear robot model. It is therefore not necessary to linearize it around the equilibrium $e_y = e_\psi = 0$ as it is classically done, for example in [11].

It also has the advantage of being independent of speed. As a result, speed variations do not affect the controller's error-correction performance. Easily deployable, its expression is quite simple, and its implementation requires only positioning information.

For a unicycle-type robot, kinematic equations of motion with the assumption of rolling without slipping, meaning a zero lateral speed and a longitudinal speed equal to the angular speed of the wheels multiplied by their radius, are as follows:

$$\underbrace{\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix}}_{\dot{q}} = \begin{bmatrix} u_1 \cos\psi \\ u_1 \sin\psi \\ u_2 \end{bmatrix} \tag{3.1}$$

with u_1 the longitudinal velocity and u_2 the yaw rate inputs.

Let us consider a reference frame F_r that the robot must pursue in position and orientation, defined by the smooth time function (x_r, y_r, ψ_r) which is solution to the robot's kinematic model for a specific reference control input $u_r = (u_{1,r}, u_{2,r})$. Then, note (e_x, e_y, e_ψ) the tracking error in position, meaning the error between the physical robot and the reference robot.

As described in [77] by using the Lyapunov stability theorem, the following feedback nonlinear control input is proved to asymptotically stabilize the tracking error at zero:

$$\begin{cases} u_1 = (w_1 + u_{1,r})/\cos(e_\psi) \\ u_2 = w_2 \cos^2(e_\psi) + u_{2,r} \end{cases}$$

by using the following control law

$$\begin{cases} w_1 = k_1 i_1 \\ w_2 = k_2 i_2 + k_3 i_3 \end{cases}$$

with

$$\begin{cases} i_1 = -|u_{1,r}|(e_x + e_y \tan(e_\psi)) \\ i_2 = -u_{1,r} e_y \\ i_3 = -|u_{1,r}| \tan(e_\psi) \end{cases}$$

This controller is established by changing coordinates and control variables to express the control equations as a chained system. It is valid only if the orientation error e_ψ remains strictly less than $\pi/2$.

While this controller has the advantage of easy implementation, it is only effective in application cases without slippage. Designed for slip-free movement on horizontal terrain, it does not consider the consequences of high dynamic solicitations on the lateral behavior of the robot. However, mobile robots can operate on floors with low grip, such as carpets. Another sliding factor is the robot load. Depending on whether the robot is navigating empty or carrying a large load, its adhesion conditions change. Finally, low-level control and actuators response time can have a significant impact on the quality of path tracking.

For all these reasons, the first simulation evaluations, reported in section 3.4, were not very conclusive. Significant tracking errors are observed which in real life situations with real robots can lead to erratic behavior.

Under these conditions, a new control strategy that takes into account the dynamics of the system and the information provided by the motion planning is necessary.

In the next section, based on a predictive control law [39], we will adapt it to the model of our robot and modify it to take into account the reference trajectory information on the prediction horizon already provided by our planner. This nonlinear controller is based on the dynamic model of the robot. In particular, it takes into account the inertial properties of the system.

3.3 Modification of a Nonlinear Continuous Generalized Predictive Control

Including a model predictive controller results in the architecture presented in Fig. 3.1 in the introduction of this chapter for each robot.

The authors of [39] propose a Nonlinear Continuous-time Generalized Predictive Control (NCGPC) meant for outdoor mobile robots. Following their approach, we derive a different control law that takes advantage of our receding horizon planner. That new control law is created by replacing the approximation for the reference output used in their approach by the prediction of our motion planner DRHMP.

3.3.1 Extended Unicycle Model

To apply the same approach as in [39] we need an extended model for the unicycle mobile robot that integrates the kinematic and dynamic models. Furthermore, we need to be able to write the model in a nonlinear control-affine form as shown below in Eq. (3.2):

$$\dot{q} = f(q,u) = f_a(q) + \sum_{j=1}^{p} f_{b,j}\, u_j \tag{3.2}$$

From [19] we can write such an extended model as in Eq. (3.3):

$$\underbrace{\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{v} \\ \dot{\omega} \end{bmatrix}}_{\dot{q}} = \underbrace{\begin{bmatrix} v\cos\psi \\ v\sin\psi \\ \omega \\ \frac{\theta_3}{\theta_1}\omega^2 - \frac{\theta_4}{\theta_1}v \\ -\frac{\theta_5}{\theta_2}v\omega - \frac{\theta_6}{\theta_2}\omega \end{bmatrix}}_{f_a(q)} + \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{\theta_1} & 0 \\ 0 & \frac{1}{\theta_2} \end{bmatrix}}_{f_b=[f_{b,1}\ f_{b,2}]} \underbrace{\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}}_{u} \tag{3.3}$$

where $q \in Q \subset \mathbb{R}^n \mid n = 5$ is the state vector composed of the longitudinal and lateral positions, the orientation, and the longitudinal and angular speeds. And $u \in U \subset \mathbb{R}^p \mid p = 2$ is the system input composed of u_1 the desired longitudinal velocity and u_2 the desired yaw rate. This model is obtained by assuming a low-level PD velocity controller to create a relationship between the acceleration and the desired speed. Also, the slip speed of the drive wheels and forces on caster wheels are neglected.

The parameters vector $[\theta_1\ \theta_2\ \theta_3\ \theta_4\ \theta_5\ \theta_6]^T = \theta$ characterizing the dynamics of the robot can be determined either by system identification or by the properties of the unicycle robot, such as its mass, moment of inertia, and impedance of motors. Details on the latter method

are provided in [19]. For our particular simulated case, an identification algorithm was used based on the minimization of error in velocities followed by the minimization of error in position.

This identification process was used to find θ for both simulated and real robots. First, the position, velocity and control input of a robot were recorded during a short period of time (less than one minute). Then, the SLSPQ optimizer was used to find the set of control variables (θ) that minimize the errors in velocity and position. At each iteration of the optimizer, the dynamic model based on the current values of θ was used to simulate the velocity and position of the robot subject to the same control inputs as the real system. Fig. 3.2 shows a comparison between the real system and the identified dynamic model after convergence of the optimizer.

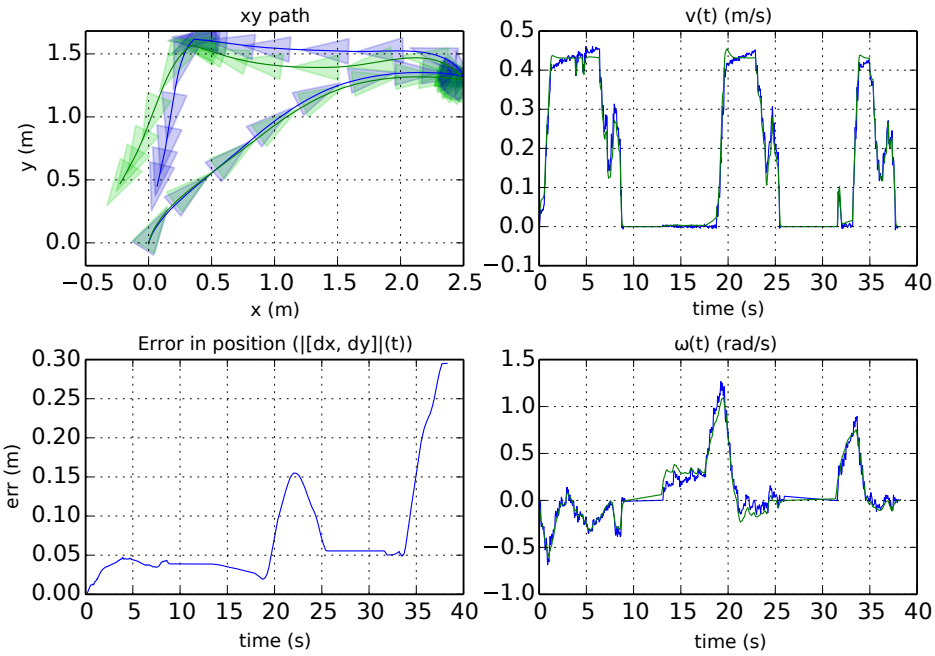


Fig. 3.2 Given the same input values ($u(t)$), the blue velocities and position represent a real turtlebot whereas the green ones are from the identified dynamic model

The discrete form with a simple integration of explicit type is given in Eq. (3.4) (the time step $t_{k+1} - t_k$ needs to be small enough, otherwise an integration of implicit type would be necessary):

$$\underbrace{\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \psi_{k+1} \\ v_{k+1} \\ \omega_{k+1} \end{bmatrix}}_{q_{k+1}} = \begin{bmatrix} v_k \cos\psi_k \\ v_k \sin\psi_k \\ \omega_k \\ \frac{\theta_3}{\theta_1}\omega_k^2 - \frac{\theta_4}{\theta_1}v_k + \frac{u_{1,k}}{\theta_1} \\ -\frac{\theta_5}{\theta_2}v_k\omega_k - \frac{\theta_6}{\theta_2}\omega_k + \frac{u_{2,k}}{\theta_2} \end{bmatrix} (t_{k+1} - t_k) + \underbrace{\begin{bmatrix} x_k \\ y_k \\ \psi_k \\ v_k \\ \omega_k \end{bmatrix}}_{q_k} \tag{3.4}$$

Written otherwise, the vector θ can then be identified from the following equation:

$$\begin{bmatrix} \frac{v_{k+1}-v_k}{t_{k+1}-t_k} & 0 & -\omega_k^2 & v_k & 0 & 0 \\ 0 & \frac{\omega_{k+1}-\omega_k}{t_{k+1}-t_k} & 0 & 0 & v_k\omega_k & \omega_k \end{bmatrix} \theta = \begin{bmatrix} u_{1,k} \\ u_{2,k} \end{bmatrix}$$

3.3.2 Cost to Minimize

The objective is to synthesize a control law that minimizes the quadratic error in position and orientation (i.e. pose) over a time-horizon ahead of the current instant t.

Since only error in pose is to be minimized, the system output can be written as follows:

$$z(t) = h(q(t)) = \begin{bmatrix} x & y & \psi \end{bmatrix}^T$$

with $z \in Z \subset \mathbb{R}^m \mid m = 3$. And the error as:

$$e(t) = z(t) - z_{\text{ref}}(t)$$

where $z_{\text{ref}}(t)$ is the reference output provided by the motion planning.

The criterion to be minimized can be written as:

$$J = \sum_{i=1}^{m} \tfrac{1}{2} \int_0^{T_i} (e_i(t+\tau))^2 d\tau$$

where T_i is the prediction horizon for the ith element of $z(t)$ and $e_i(t+\tau)$ the ith element of the prediction error at $t+\tau$ with $0 < \tau \leq T_i$. In this particular case, to find the control law that minimizes J is to find u satisfying the equation:

$$\frac{\partial J}{\partial u} = 0_{p\times 1}$$

For solving the above equation, an expression for the prediction error must be defined and the criterion rewritten in a matrix form.

3.3.3 Predictive Error Definition

In order to obtain an equation of error $e(t+\tau)$ where the system input u is explicitly present, we rewrite $z(t+\tau)$ using Taylor series:

$$z_i(t+\tau) = \sum_{k=0}^{\rho_i} z_i^{(k)}(t)\frac{\tau^k}{k!} + \epsilon(\tau^{\rho_i})$$

As explained in [39] the vector $\rho = [\rho_1 \cdots \rho_m]$ is the relative degrees of a nonlinear MIMO system (Multiple Input Multiple Output). It is a vector composed of possibly different values of relative degrees ρ_i for each output z_i. ρ_i is the least number of derivatives required to make explicit in the expression of z_i at least one component of the input vector u.

Furthermore, a nonlinear control-affine MIMO system (Eq. (3.2)) has a relative degree $\rho = [\rho_1 \cdots \rho_m]$ around q^0 if:

1. $L_{f_{b,j}} L_{f_a}^{(k)} z_i = 0$ for all $1 \leq j \leq p$, for all $k < \rho_i - 1$, for all $1 \leq i \leq m$ and for all q in the neighborhood of q^0

2. the product $D^T D$ is non-singular, D being the decoupling matrix of dimension $m \times p$, given by:

$$D = \begin{bmatrix} L_{f_{b,1}} L_{f_a}^{(\rho_1-1)} z_1 & \cdots & L_{f_{b,p}} L_{f_a}^{(\rho_1-1)} z_1 \\ \vdots & \ddots & \vdots \\ L_{f_{b,1}} L_{f_a}^{(\rho_m-1)} z_m & \cdots & L_{f_{b,p}} L_{f_a}^{(\rho_m-1)} z_m \end{bmatrix} \tag{3.5}$$

Here above, the Lie derivative of the output function z_i along f, in $q \in \mathbb{R}^n$ is defined as follows:

$$L_f z_i(q) = \sum_{j=1}^{n} \frac{\partial z_i}{\partial q_j}(q) f_j(q)$$

with $1 \leq j \leq p$ and $1 \leq i \leq m$ ($p = 2$; $m = 3$). For these values of i, we note that $f_1 = f_{a1}$ and $f_2 = f_{a2}$, thus $f_i = f_{ai}$. Then, we use the standard geometric notation for Lie derivatives summarized below by its recursive expression:

$$\begin{cases} L_f^{(0)} z_i &= z_i \\ L_f^{(k)} z_i &= L_f L_f^{(k-1)} z_i = \frac{\partial L_f^{(k-1)} z_i}{\partial q} f = z_i^{(k)} \end{cases}$$

Using this notation, $L_{f_{b,j}} L_{f_a}^{(k)} z_i$ in condition 1) can be read as the Lie derivative of the kth Lie derivative of z_i with respect to f_a with respect to $f_{b,j}$.

Rewriting the expression for $z_i(t+\tau)$ in a matrix form and excluding the remainder term, we obtain the following approximation:

$$z_i(t+\tau) \simeq \underbrace{\begin{bmatrix} 1 & \tau & \cdots & \frac{\tau^{\rho_i}}{\rho_i!} \end{bmatrix}}_{\Lambda_i} \begin{bmatrix} z_i(t) & \dot{z}_i(t) & \cdots & z_i^{(\rho_i)}(t) \end{bmatrix}^T$$

Replacing the first matrix by the more compact notation Λ_i and using Lie derivatives, one can write:

$$z_i(t+\tau) \simeq \Lambda_i L_{z_i} \tag{3.6}$$

where

$$L_{z_i} = \begin{bmatrix} L_f^{(0)} z_i(t) & L_f^{(1)} z_i(t) & \cdots & L_f^{(\rho_i-1)} z_i(t) & L_f^{(\rho_i)} z_i(t) \end{bmatrix}^T$$

which, assuming condition 1) above, can be simplified (all lines of L_{z_i} except the last one) to:

$$L_{z_i} = \begin{bmatrix} L_{f_a}^{(0)} z_i(t) \\ \vdots \\ L_{f_a}^{(\rho_i-1)} z_i(t) \\ L_{f_a}^{(\rho_i)} z_i(t) + L_{f_b}(L_{f_a}^{(\rho_i-1)} z_i(t)) u(t) \end{bmatrix}$$

This last form of L_{z_i} makes the system input u explicit in the expression of $z_i(t+\tau)$. Functions f, f_a and f_b come from the model in Eq. (3.2).

As for the second term in the prediction error expression, $z_{\text{i,ref}}(t+\tau)$, it can be kept undetermined until the expression for u, meaning the control law, is found. This is possible because our planner can give its value for any $\tau \mid 0 < \tau \leq T_i$ as long as $T_i \leq T_p - T_c$. That last condition over T_i is needed because $z_{\text{ref}}(t + T_p - T_c)$ is the further in time the planner can output a valid reference trajectory for any given t.

3.3.4 Control Law equation

After some algebraic manipulation, we derive the final expression for the control law as shown in Eq. (3.7).

$$\begin{aligned} &\frac{\partial J}{\partial u} = 0_{p\times 1} \\ &\quad \Rightarrow u = -(D^T D)^{-1} D^T (K^{ss})^{-1} (K^s L_z - R^s) \end{aligned} \tag{3.7}$$

where D is the decoupling matrix (Eq. (3.5)), K^{ss} and K^s the gain matrices (Eq. (3.8) to (3.10)), L_z the prediction output matrix (Eq. (3.11)) and R^s the future reference output matrix

(Eq. (3.12) and (3.13)). The detailed steps to arrive at Eq. (3.7) are showed in Appendix B.

$$K^s = \text{diag}([K_1^s \ \cdots \ K_m^s])K^{ss} = \text{diag}([K_1^{ss} \ \cdots \ K_m^{ss}]) \tag{3.8}$$

with K_i^s the last line of the matrix K_i and K_i^{ss} the last element of the vector K_i^s.
K_i is defined as:

$$K_i = \int_0^{T_i} \Lambda_i^T \Lambda_i d\tau \tag{3.9}$$

which gives the following expression for each element of K_i:

$$K_{i,(a,b)} = \frac{T_i^{(a+b)+1}}{((a+b)+1)a!b!} \tag{3.10}$$

with $a,\ b \in [0,\ \rho_i] \subset \mathbb{Z}$ the row and column indexes.

$$L_z = \begin{bmatrix} z_1 & \cdots & L_{f_a}^{(\rho_1)} z_1 & \cdots & z_m & \cdots & L_{f_a}^{(\rho_m)} z_m \end{bmatrix}^T \tag{3.11}$$

$$R^s = \begin{bmatrix} R_1^s & \cdots & R_m^s \end{bmatrix}^T \tag{3.12}$$

R_i^s is the last element of the row vector R_i defined as:

$$R_i = \int_0^{T_i} z_{i,\text{ref}} \Lambda_i d\tau \tag{3.13}$$

Now that the general expression of the control input u is defined, we can specialize it for our particular case of a unicycle robot represented by the model in Eq. (3.3).

3.3.5 Control Law Equation for a Unicycle-like Vehicle

In order to find u, matrices D, K^s, K^{ss}, L_y and R^s must be determined. To do so, the vector ρ is needed. A way of finding ρ for our particular unicycle system is by computing $L_{f_{b,j}} L_{f_a}^{(k)} y_i$ for k beginning at 0, and incrementing it until the conditions 1) and 2) presented before are satisfied.

For $\rho = [1\ 1\ 1]$, $L_{f_{b,j}} L_{f_a}^{(0)} y_i = L_{f_{b,j}} y_i = 0$ for all $1 \leq j \leq p$, for all $1 \leq i \leq m$ which does not satisfy the second condition as shown below:

$$\begin{cases} L_{f_{b,1}} y_1 &= [1\,0\,0\,0\,0\,0][0\,0\,0\,1/\theta_1\,0]^T = 0 \\ L_{f_{b,1}} y_2 &= [0\,1\,0\,0\,0\,0][0\,0\,0\,1/\theta_1\,0]^T = 0 \\ L_{f_{b,1}} y_3 &= [0\,0\,1\,0\,0\,0][0\,0\,0\,1/\theta_1\,0]^T = 0 \end{cases}$$

$$\begin{cases} L_{g_2} y_1 &= [1\,0\,0\,0\,0\,0][0\,0\,0\,0\,1/\theta_2]^T = 0 \\ L_{g_2} y_2 &= [0\,1\,0\,0\,0\,0][0\,0\,0\,0\,1/\theta_2]^T = 0 \\ L_{g_2} y_3 &= [0\,0\,1\,0\,0\,0][0\,0\,0\,0\,1/\theta_2]^T = 0 \end{cases}$$

For computing $L_{f_{b,j}} L_{f_a}^{(k)} y_i$ with $\rho = [2\ 2\ 2]$ we need first $L_{f_a} y_i$:

$$\begin{cases} L_{f_a} y_1 &= [1\,0\,0\,0\,0\,0] f_a(q) = v\cos\psi \\ L_{f_a} y_2 &= [0\,1\,0\,0\,0\,0] f_a(q) = v\sin\psi \\ L_{f_a} y_3 &= [0\,0\,1\,0\,0\,0] f_a(q) = \omega \end{cases}$$

Computing now $L_{f_{b,j}} L_f y_i$ we obtain:

$$\begin{cases} L_{f_{b,1}} L_{f_a} y_1 &= [0\,0\,-v\sin\psi\ \cos\psi\ 0\,0] f_{b,1}(q) = \cos\psi/\theta_1 \\ L_{f_{b,1}} L_{f_a} y_2 &= [0\,0\,v\cos\psi\ \sin\psi\ 0\,0] f_{b,1}(q) = \sin\psi/\theta_1 \\ L_{f_{b,1}} L_{f_a} y_3 &= [0\,0\,0\,0\,0\,1] f_{b,1}(q) = 0 \end{cases}$$

$$\begin{cases} L_{f_{b,2}} L_{f_a} y_1 &= [0\,0\,-v\sin\psi\ \cos\psi\ 0\,0] f_{b,2}(q) = 0 \\ L_{f_{b,2}} L_{f_a} y_2 &= [0\,0\,v\cos\psi\ \sin\psi\ 0\,0] f_{b,2}(q) = 0 \\ L_{f_{b,2}} L_{f_a} y_3 &= [0\,0\,0\,0\,0\,1] f_{b,2}(q) = 1/\theta_2 \end{cases}$$

which gives the following decoupling matrix:

$$D = \begin{bmatrix} \frac{\cos\psi}{\theta_1} & 0 \\ \frac{\sin\psi}{\theta_1} & 0 \\ 0 & \frac{1}{\theta_2} \end{bmatrix}$$

and consequently:

$$D^T D = \begin{bmatrix} \frac{1}{\theta_1^2} & 0 \\ 0 & \frac{1}{\theta_2^2} \end{bmatrix}$$

which is non-singular for all $\theta_1, \theta_2 \neq 0$. Besides, the first condition is also met: $L_{f_{b,j}} L_{f_a}^{(k_i)} y_i = 0\ \forall\ k_i < \rho_i - 1$.

$\rho = [2\ 2\ 2]$ is then a solution. Consequently, matrices K^s, K^{ss} and L_y can be written as below:

$$K^{ss} = \mathrm{diag}\left(\begin{bmatrix} \frac{T_1^5}{20} & \frac{T_2^5}{20} & \frac{T_3^5}{20} \end{bmatrix}\right)$$

$$K^s = \mathrm{diag}\left(\left[\begin{bmatrix} \frac{T_1^3}{6} & \frac{T_1^4}{8} & \frac{T_1^5}{20} \end{bmatrix} \cdots \begin{bmatrix} \frac{T_3^3}{6} & \frac{T_3^4}{8} & \frac{T_3^5}{20} \end{bmatrix}\right]\right)$$

$$L_z = \begin{bmatrix} x \\ v\cos\psi \\ \left(\frac{\theta_3}{\theta_1}\omega^2 - \frac{\theta_4}{\theta_1}v\right)\cos\psi - v\omega\sin\psi \\ y \\ v\sin\psi \\ \left(\frac{\theta_3}{\theta_1}\omega^2 - \frac{\theta_4}{\theta_1}v\right)\sin\psi + v\omega\cos\psi \\ \psi \\ \omega \\ -\frac{\theta_5}{\theta_2}v\omega - \frac{\theta_6}{\theta_2}\omega \end{bmatrix}$$

R^s can be found (numerically or analytically) from the planner's output according to the following expression:

$$R^s = \frac{1}{2}\begin{bmatrix} \int_0^{T_1} x_{\text{ref}}(t+\tau)\tau^2 d\tau \\ \int_0^{T_2} y_{\text{ref}}(t+\tau)\tau^2 d\tau \\ \int_0^{T_3} \psi_{\text{ref}}(t+\tau)\tau^2 d\tau \end{bmatrix} \tag{3.14}$$

Thus the complete expression for u becomes:

$$\begin{aligned} u \;=\; & -10\begin{bmatrix} \theta_1/T_1^5\cos\psi & \theta_1/T_2^5\sin\psi & 0 \\ 0 & 0 & \theta_2/T_3^5 \end{bmatrix} \\ & \left(\frac{1}{60}\begin{bmatrix} 20T_1^3x + 15T_1^4 v\cos\psi + 6T_1^5\left(\left(\frac{\theta_3}{\theta_1}\omega^2 - \frac{\theta_4}{\theta_1}v\right)\cos\psi - v\omega\sin\psi\right) \\ 20T_2^3y + 15T_2^4 v\sin\psi + 6T_3^5\left(\left(\frac{\theta_3}{\theta_1}\omega^2 - \frac{\theta_4}{\theta_1}v\right)\sin\psi + v\omega\cos\psi\right) \\ 20T_3^3\psi + 15T_3^4\omega + 6T_3^5\left(-\frac{\theta_5}{\theta_2}v\omega - \frac{\theta_6}{\theta_2}\omega\right) \end{bmatrix}\right. \\ & \left. - \begin{bmatrix} \int_0^{T_1} x_{\text{ref}}(t+\tau)\tau^2 d\tau \\ \int_0^{T_2} y_{\text{ref}}(t+\tau)\tau^2 d\tau \\ \int_0^{T_3} \psi_{\text{ref}}(t+\tau)\tau^2 d\tau \end{bmatrix}\right) \end{aligned} \tag{3.15}$$

3.3.6 Desired Trajectory Definition from DRHMP Solution

Because the equation for u_2 (angular velocity control input) does not show any terms using the error in position (x, y), the input cannot correct errors in position if no error in velocity nor orientation is present.

To solve that problem, we derive new desired input values x_{desired}, y_{desired}, ψ_{desired} and use them instead of x_{ref}, y_{ref}, ψ_{ref} in Eq. (3.15). The desired values are chosen so lateral position error can be corrected and the vehicle stabilized around the planner's trajectory even when only that kind of error is present. The position error is as follows: $e_{\text{XY}} = [x_{\text{ref}} - x,\ y_{\text{ref}} - y]^T$.

That error in the vehicles frame of reference is then $\vec{l} = \mathrm{R}_{(-\psi)} e_{\mathrm{XY}}$ and the lateral error is the second element of $\vec{l}$, l_1. $\mathrm{R}_{(-\psi)}$ represents the 2D rotation matrix of the vehicle's negative yaw $(-\psi)$.

Two approaches were then considered:

1. Use x_{desired}, y_{desired}, ψ_{desired} equals x_{ref}, y_{ref}, $\psi_{\mathrm{ref}} + \arctan(\alpha l_1)$ with

$$\alpha = c\left(1 - \frac{\arctan\left(s\left(2\tau / T_3 - 1\right)\right) + \arctan(s)}{2\arctan\left(s\right)}\right)$$

   where c is a positive value behaving as a gain (a convergence factor), giving the span of α ($\alpha \in [0, c]$). Constant s is defined over $[0, \infty[$, and it is a shape factor; the smaller it is, the closer to a straight line α is, conversely, the bigger the s, the closer to a step function. τ is the time variable used for computing R^s.

   α is inspired by sigmoid functions such as the error function (erf) and can be seen as a model for predicting how the lateral error will decrease with time.

   Fig. 3.3 shows curves $\alpha(\eta)/c \mid \eta = \tau / T_3$ for different s constants to help visualize the type of curve used.

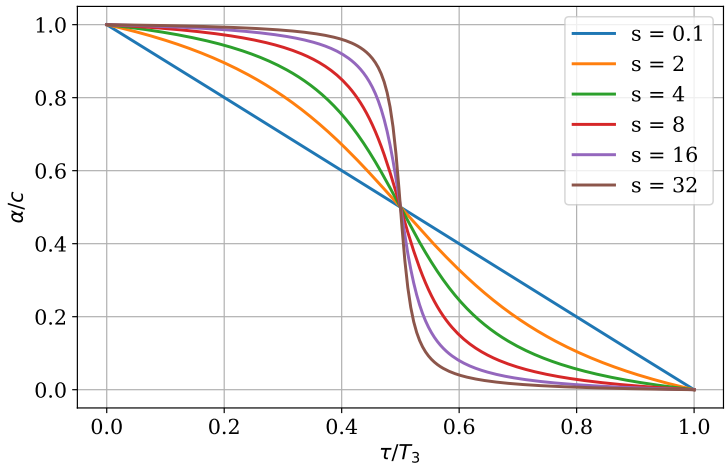


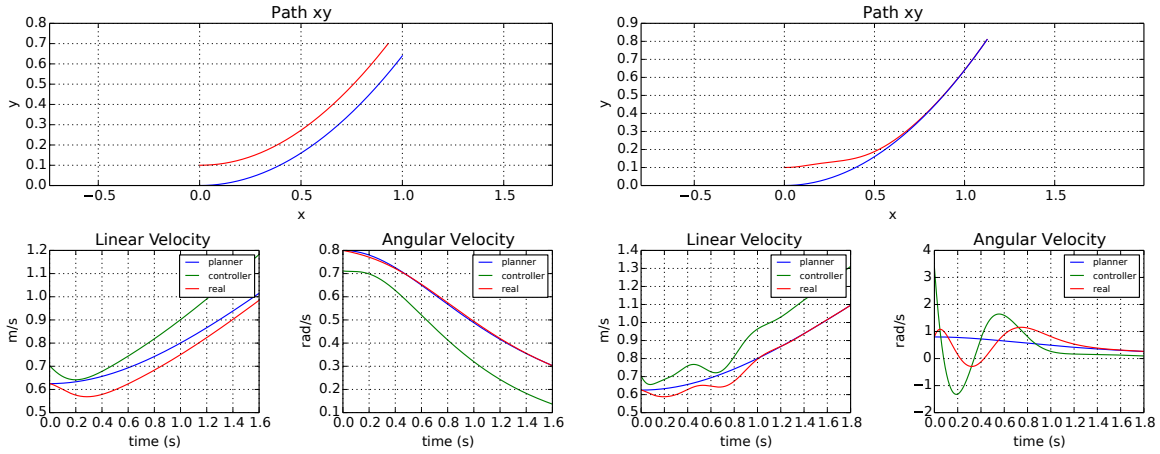
Fig. 3.3 Representation of $\alpha(\tau)$ fuction for different values of s

2. Change the B-spline control points defining the reference trajectory generating then the desired values input values. The new control points would be generated as explained below:

$$\hat{C}_i = C_i - \alpha_i e_{\mathrm{XY}}$$

$$\alpha_i = c\left(1 - \frac{\arctan\left(s\left(2d_i/D - 1\right)\right) + \arctan(s)}{2\arctan\left(s\right)}\right)$$

with D equals the B-spline control polygon length and d_i the sum of distances between successive control points from C_0 to C_i. s, c are analogous to the previous case.



(a) (b)

Fig. 3.4 Simulation of the NCGPC-M controller without (a) and with (b) desired reference correction based on the correction approach (1) using $s = 2$ and $c = 1$

In our experiments, we used the first approach for generating the desired values. Fig. 3.4 shows a simulation where the problem of applying the reference trajectory directly is present. The result when the correction from approach (1) is applied is shown as well in Fig. 3.4b. On Fig. 3.4, the red xy paths (on the two top plots) represent the controlled system position, while the blue path is the reference xy path. Similarly, blue linear and angular velocities on plots in the bottom are the reference values, red is the actual system's velocities and green is the NCGPC-M output (the system control input).

Given the satisfactory performance of the approach (1), the second approach, based on changing the control points, has not been tested.

3.4 Experimental Evaluation

We compared the different control approaches using dynamic simulations with XDE. XDE is a simulation engine developed at CEA [57]. Its visual environment can be seen in Fig. 3.5.

The simulated robots were set to weigh 55 Kg, an overall shape similar to that of an Adept Lynx mobile platform [63], the two drive wheels were placed inline with the geometric center

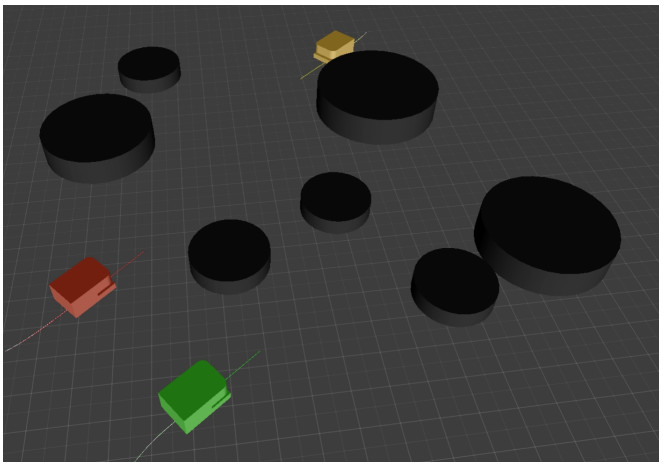


Fig. 3.5 XDE 3D visual environment where 3 robots navigate among 7 obstacles

of the robot and the track length was around 20 cm. They had a radius of near 10 cm and had material properties close to rubber. Four caster wheels were added to each corner of the robot.

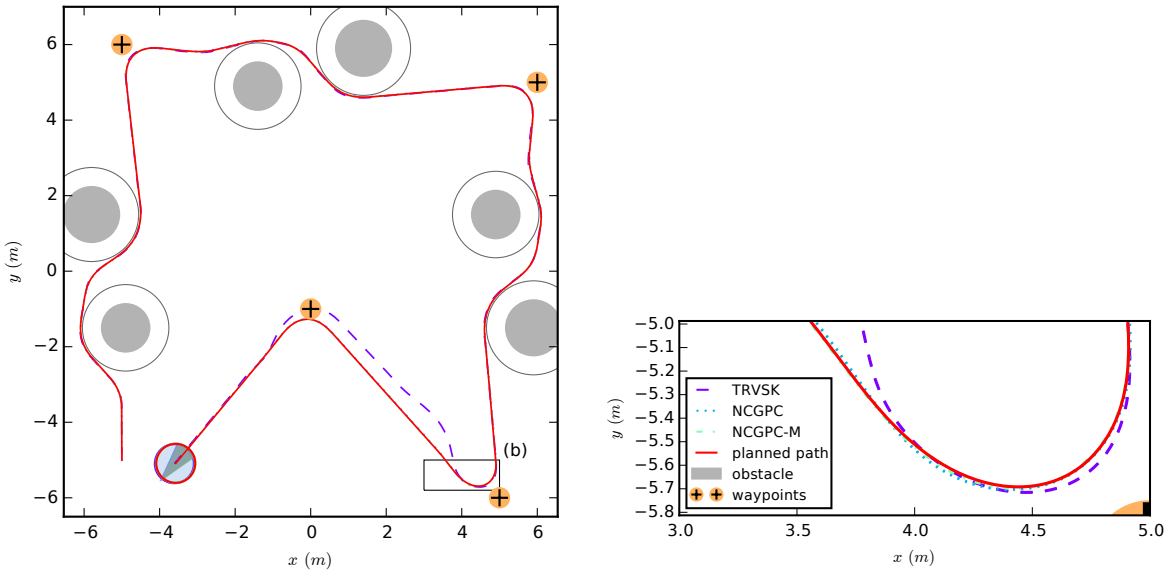
In order to analyze the controller performance, three different control laws were compared: **NCGPC** (Non-linear Continuous-time Generalized Predictive Control) is the initial control law presented in [39]; **NCGPC-M** (NCGPC-Modified) is the control law presented in the previous section 3.3; **TRVSK** (Tracking Reference Vehicle with Same Kinematics) introduced in [77] is discussed in previous section 3.2 and, differently from the other two control laws, it is not predictive.

NCGPC-M and NCGPC differ on how they take x_{desired}, y_{desired}, ψ_{desired} into account. NCGPC equation for the control input u results from an extrapolation of the reference output at instant τ, forward in time. Meanwhile NCGPC-M, by avoiding that extrapolation, has the matrix R^s in its expression for u which integrates $x_{\text{desired}}(t)$, $y_{\text{desired}}(t)$, $\psi_{\text{desired}}(t)$ over time.

Table 3.1 Comparison of control laws

	TRVSK	NCGPC	NCGPC-M
RMS($\|[x_{\text{err}}\ y_{\text{err}}]\|$) (cm)	6.93	1.17	0.44
max($\|[x_{\text{err}}\ y_{\text{err}}]\|$) (cm)	31.28	4.26	1.92
RMS(ψ_{err}) (deg)	2.78	0.75	0.34
max(ψ_{err}) (deg)	16.29	4.84	1.28

Fig. 3.6 shows the result of three identical simulations (same reference trajectory, robot, obstacles) except for the control law adopted to follow the reference trajectory. 6 obstacles



(a) (b)

Fig. 3.6 Control laws comparison. (a) General configuration of the simulation. (b) Zoom on the robots' paths stressing the non-coincidence of the planned path and the three executed paths for each control law. Travel time is around 48 s.

were placed in the simulated area as well as 4 waypoints to which the robot passed by before reaching its goal near point $(-4,-5)$. Motion planning main parameters, T_p and T_c, were set to 1.2 s and 0.3 s respectively[1]. The three different paths for each simulation and the reference trajectory are overlapped, and their non-coincidence can better be seen in Fig. 3.6b. Table 3.1 shows a comparison of the three control laws based on the results of the three simulations described in Fig. 3.6. Additionally, Fig. 3.7 shows the pose error during the first 20 seconds of the simulations, which is nearly the first half of the robot's path[2].

From Table 3.1 and Fig. 3.7, we can see that NCGPC-M shows the smaller root mean square (RMS) and smaller maximum values for both position and orientation errors. This indicates that the NCGPC-M is the control law that performs the best among the three studied with regard to error minimization.

Table 3.2 shows the mean and standard deviation of 4778 measurements of elapsed time for each of the four routines: the output evaluation routine in the planner and the three different control routines. All controllers were coded in C++03 STL language and compiled using Visual C++ 10.0 compiler. They were run on an Intel i7-5600U CPU. TRVSK and NCGPC approaches were implemented having constant time complexity while

[1] For some discussion about choosing values for T_p and T_c see [53]

[2] in the second half of the path, some high differences between errors make difficult to appreciate the graph

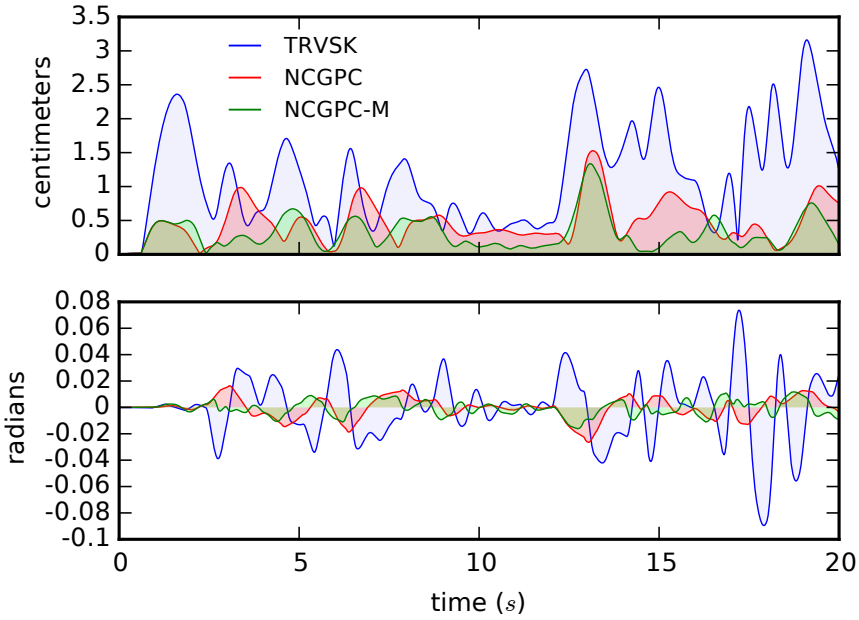


Fig. 3.7 Errors in position and orientation for the first 20 seconds of the simulation shown in Fig. 3.6.

NCGPC-M implementation was $O(n)$ on the number of samples used for integration when approximating matrix R^s (Eq. (3.14)). If an analytical solution for R^s would be provided, NCGPC-M could also have constant complexity, but it would not necessarily be quicker than the current implementation for a relatively small number of samples.

Table 3.2 Performance of planning and control implementations on an Intel i7-5600U CPU

	TRVSK	NCGPC	NCGPC-M
Mean elapsed time (μs)	1.79	1.89	33.86
Standard deviation (μs)	1.12	0.59	6.09

The NCGPC-M control, which had the best performance for minimizing the position and orientation errors, presents the highest computational cost of the three controllers, which is expected since it computes a numerical integration the others do not. In Fig. 3.8 we see a histogram based on the same 4778 calls to the NCGPC-M controller. For 95.44% of the calls the elapsed time was inferior to 35 μs (~29 kHz) and in no case the elapsed time was bigger than 140 μs (~7 kHz). These performances therefore makes this controller suitable for a realtime implementation.

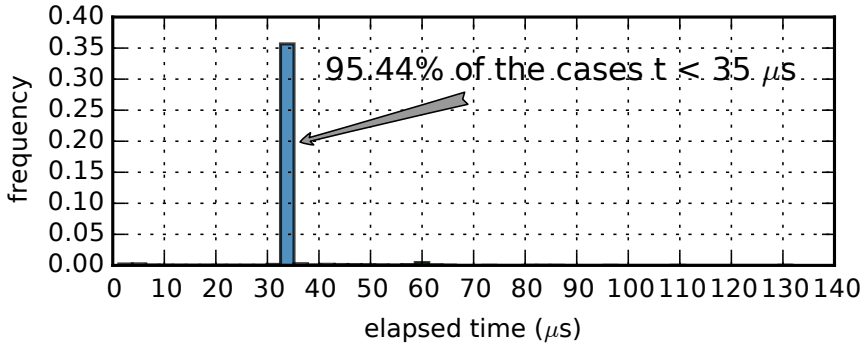


Fig. 3.8 Histogram of controller frequency. Plot based on 4778 calls to the NCGPC-M controller

3.5 Conclusions

We generalized a Nonlinear Continuous Generalized Predictive Control (NCGPC) for taking complete parameterized trajectories instead of estimating the future reference output by extrapolating its current value, thus taking advantage of our trajectory planning approach and improving the controller performance.

The simulated results suggest that this approach allows a system of multiple unicycle-like robots to navigate collision-free with errors from planned position below 2 cm. Furthermore, the results indicate a response frequency for the planner and controller higher than 2 kHz what would allow for their use in a real-time system.

Chapter 4

Integrated Task and Motion Planning

4.1 Introduction

In the previous chapters, we have proposed a local receding horizon planning algorithm and a control law able to take dynamics effects into account. In an efficient multi-robot system, there remains the problem of allocating tasks to different robots and deriving sequence of actions (*plans*) that can carry out those tasks. This problem is particularly difficult in a partially known environment.

A robotic system for real-life applications needs to address two sources of uncertainties when planning tasks: the first one about the world state (*sensing uncertainty*) and the second about the outcome of actions (*predictability uncertainty*). From a decision-theoretic point of view, this problem is simply described as **planning under uncertainty**. The typical scientific and industrial use cases for which we developed the approaches in chapters 2 and 3 of this work present great amounts of both sources of uncertainties as most real scenarios do.

A possible way of modeling planning under uncertainty problems is by using Partially Observable Markov Decision Processes (POMDPs). POMDP models a decision process in which it is assumed that outcomes of actions are partly random and partly under the control of a decision maker (an agent), but the agent cannot directly observe the underlying world state. Instead, it must maintain a probability distribution over the set of possible states, based on a set of observations and observation probabilities, and the underlying Markov Decision Process (MDP). Solving POMDPs in real scenarios is usually intractable as discussed in [40], thus many approaches have been proposed to reduce the complexity of the problem.

Hierarchical approaches to planning, introduced more than 40 years ago [74], have been considered for trying to solve these difficult problems by exploiting some hierarchical knowledge about the planning domain. Among the five works on task planning considered

in the rest of this section, three can be considered to fit this category [5, 47, 33]. Work presented in [80] is a generic framework that could in theory apply different techniques for task planning, including hierarchical ones. Only work in [66] avoids completely hierarchical knowledge by planning in a higher abstraction level with planning blocks called skills.

In particular, work presented in [5] uses a variant of decentralized POMDPs (DecPOMDPs) called MacDecPOMDPs to generate off-line policies in a warehouse domain for 3 robots. MacDecPOMDPs employs *macro-actions*: temporally extended actions which may require different amounts of time to execute to perform planning. Its planner is capable of generating cooperative behavior for complex multi-robot domains with task allocation, direct communication, and signaling behavior emerging automatically as properties of the solution. However, the time required for such an approach is still prohibitively high; computation of the policies for 3 robots takes around 1 hour.

Authors of [47] transform the POMDP into MDP by rolling the observation uncertainty into the transition model. However, solving MDPs for real case scenario remains usually untractable. The problem is overcome using techniques for extending the MDP to hierarchical tasks. They combine MAX-Q (from reinforcement learning literature) and Monte-Carlo Tree Search "intelligently" in order to solve planning in the NAMO (Navigation Among Movable Obstacles) domain. Theoretical and empirical (using simulations) analysis show that their approach is linear in the number of obstacles.

The Hierarchical Planning in the Now (HPN) approach [33] integrates task and motion planning and the same time addresses uncertainty directly. This work shares much in common with other works done for similar domains but seems to be the one better suited and most systematic for solving complex, abstract tasks in long time horizons, taking into account uncertainties and enabling information-gathering actions. They avoid trying to find optimal solutions to the underlying POMDP (which is intractable) by, broadly speaking, constructing a deterministic approximation of the dynamics, build a sequential non-branching plan, execute the plan while observing the world for changes of the expected outcomes and replan when deviations occur. Furthermore, to address the uncertainty about the current state, planning must be done in the belief space, which is the probability distributions over world states.

Work in [66] defines robots' skills as the building blocks that make up a plan. Skills are defined in a STRIPS-like manner. World model is parsed into PDDL (Planning Domain Definition Language). It overcomes uncertainties by execution monitoring and replanning. The necessary motion planning and geometric condition checks are performed within the skills, so the skills are not constrained by a specific higher-level component, thus little gain would come from a hybrid (motion and task) planning. Implementation of general robot

skills is an open problem (picking skill, for instance). Only one skill can be executed at a time (multi-threaded skill execution could potentially enable the approach to be used for multi-robot systems).

In [80], a generic algorithm for combining task and motion planning with no assumptions about their implementation is presented. It validates a high-level plan if an error-free motion plan can be found. Otherwise, the high-level state is updated with information from the partially error-free motion plan. A new task plan is generated for the updated state.

Table 4.1 Qualitative comparison between different task planning approaches

	Multi-robot	Uncertainties	Domain-independent	Scalable
Amato [5]	⋆⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆
Kaelbling [33]	⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆
Levihn [47]	⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆
Pedersen [66]	⋆	⋆	⋆⋆⋆	⋆⋆⋆⋆
Srivastava [80]	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆

As presented in the table 4.1, a qualitative comparison of the reviewed approach seems to present the HPN approach as one of the most promising regarding the management of uncertainty, its generalization, and scalability potential, but with a weakness on the management of multi-robot systems. We decided to study and work on this approach aiming to integrate it with the motion planning and control described in the previous chapters.

4.2 Basic Hierarchical Planning in the Now

As presented in the introduction, the work done in [33] proposes a promising methodology to enable real mobile robots to carry out complex tasks in real environments where uncertainty about outcomes of actions and the current state of the world are non-negligible. Their approach aims to integrate task an motion planning through a hierarchical planning architecture, and we decided to apply this method to our task planning problem, extending it for multi-robot planning and exploiting our planning algorithm at the lower level.

They address future-state uncertainty by planning in approximate deterministic models, performing execution monitoring, and replanning when necessary. On the other hand, current-state uncertainty is handled by planning in belief space: the space of probability distributions over possible underlying world states. Modeling the robot's inability to know

precisely the world state and its dynamics enables planning to combine actions that may change the world state like moving an object or collecting information such as sensing the position of an object.

We will first describe the HPN approach on an instructive example. Consider a domain as shown in figure 4.1 where the robot (constituted of a mobile base and a manipulator arm) has the goal of having object *a* cleaned by going to the washer and then stored in the storage in place of object *b*.

The HPN recursive process of planning and execution starts from the goal represented as Plan 1 at the top of Fig. 4.2 and decomposes it in subgoals, leading to a sequence of primitive actions represented in red color in the same figure. The plan is made backward, starting from the top-level goal. This is called *goal regression* or *pre-image backchaining*.

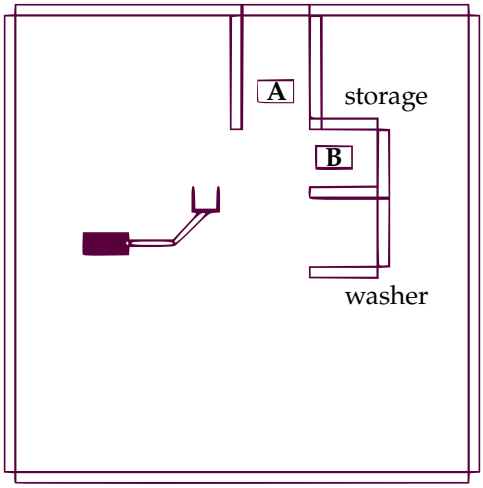


Fig. 4.1 Washing domain, in which the robot must move object *a* to the washing area, wash it, and put it in the storage area.

The HPN algorithm finds out a plan to achieve this specific top-level goal based on two abstract operations (run the washer with object *a* inside and then place it in the storage). These operations are then recursively planned and executed. If the operation is a primitive action, it is executed directly (see red nodes in Fig. 4.2), otherwise, a subgoal is built consisting of the conditions to guarantee that the other operations in the high-level plan will succeed. *Abstract operations* are created by postponing preconditions, e.g., the operation at the abstraction level 0 "run the washer with *a* inside" represented by *A0: Wash(a)* ignores the precondition that the object *a* has to be placed inside the washer if it is somewhere else in the domain.

Notice that with this approach, the built plan is such that object *b* has to be moved and placed somewhere else in order to object *a* to be placed in the storage. It is not optimal

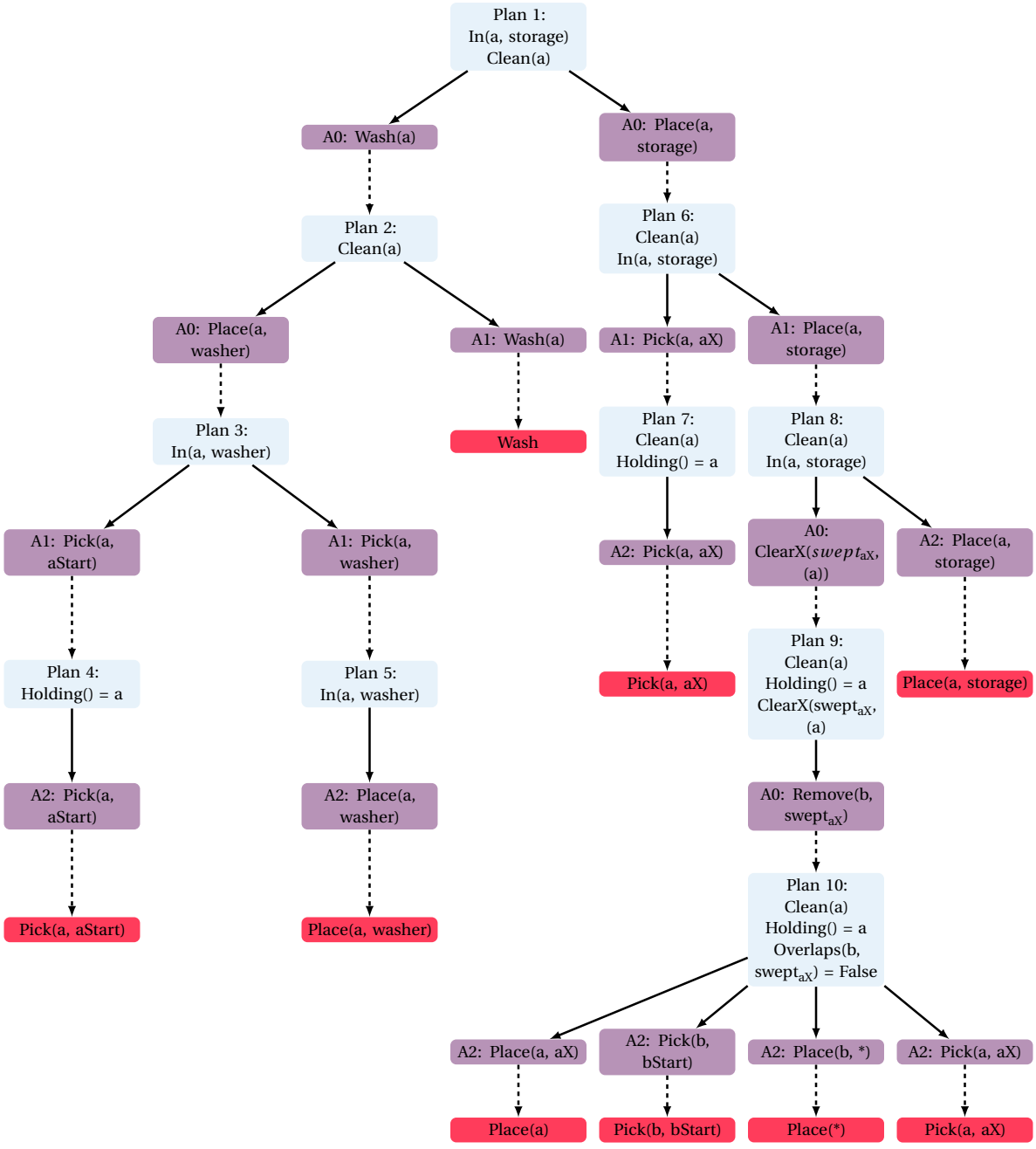


Fig. 4.2 Planning and execution tree for washing and storing an object. The wildcard character * replaces any free space where an object could be placed

though, we see that the plan goes in a way that the robot first picks up the washed object a before "realizing" that b has to be moved.

The great advantage of this approach (as for other HTN approaches) is to keep planning feasible by decomposing in time the problem of achieving a goal.

Algorithm 1 Hierarchical planning in the now

```
1:  function HPN(s_now, γ, 𝒪, α, world)
2:      if holds(γ, s_now) then
3:          return TRUE
4:      else
5:          p ← PLAN(s_now, γ, 𝒪, α)
6:          for (o_i, g_i) ∈ p do
7:              if ISPRIM(o_i) then
8:                  s_now ← world.EXECUTE(o_i)
9:              else
10:                 HPN(s_now, g_i, 𝒪, NEXTLEVEL(α, o_i), world)
11:             end if
12:         end for
13:     end if
14: end function
```

Algorithm 1 summarizes what was just described. γ represents the high level goal state for which we are planning, α is the level of abstraction of the plan which is used for relaxing preconditions of actions and creating abstract operators, s_{now} is the current world state, $\mathcal{O}$ represents the set of operators available for planning, and o_i represents the ith operator of a plan with g_i being the associated pre-image (an intermediate state or sub-goal).

4.3 Dealing with Uncertainty

As said before, in a real system, planning tasks and motions have to consider the uncertainty of outcomes of actions and the current state of the world. This does not invalidate the use of the hierarchical planning algorithm briefly presented above, but the approach needs to be extended. The next subsection shows how to adapt this planning method for handling uncertainties.

4.3.1 Modeling the Process

In order to take into account future-state uncertainty, the authors of [33] construct an approximated deterministic model of the dynamics that can be seen as a problem of finding a minimum cost path in a graph with positive weights.

Starting from the the definition of a Markov decision process (definition 1) the authors generalize it as a stochastic shortest-path problem (SSPP) (definition 2).

Definition 1 *A Markov decision process (*MDP*) is a tuple* $\langle S, A, T, R\rangle$ *where* S *is a set of world states,* A *is a set of actions,* T *is a probabilistic Markov transition model, with* $T(s,a,s') = \Pr(S_{t+1} = s' \mid S_t = s, A_t = a)$*, and* R *is a reward function where* $R(s,a)$ *is the immediate value of taking action* a *in state* s*.*

Definition 2 *A stochastic shortest-path problem (*SSPP*)* $\langle S, A, T, C, G\rangle$ *is an* MDP *in which all rewards are negative, there is a set* $G \subset S$ *of goal states, and the objective is to minimize the total expected cost (negative reward) incurred before reaching a state in* G *and terminating. We define cost function* $C(s,a)$ *in the* SSPP *to be* $-R(s,a)$ *in the original* MDP*, so that all costs are strictly positive.*

Then, they introduce the idea that choices are made by an agent and convert the SSPP into its deterministic version (DSSPP) (definition 3) in which there is a directed arc connecting two world states.

Definition 3 *A determinized* SSPP *is a tuple* $\langle S, A', W, G\rangle$ *where:* $\langle S, A, T, C, G\rangle$ *is a* SSPP*;* S *is a set of states which are nodes in a graph;* A *is a set of actions* (a, s')*, so that* $(s, a, s') \in S \times A \times S$ *is a directed arc from node* s *to node* s'*; and* W *is a weight function, so that* $W(s,a,s)$ *is the weight on arc* (s,a,s')*, which may be infinite.*

Following, in order to have paths in graph that are likely to reach a desired goal and also minimize transition cost, they define an α-cost-likelihood DSSPP (CLDSSPP) where the weight of going from a state s to s' taking the action a is as in definition 4.

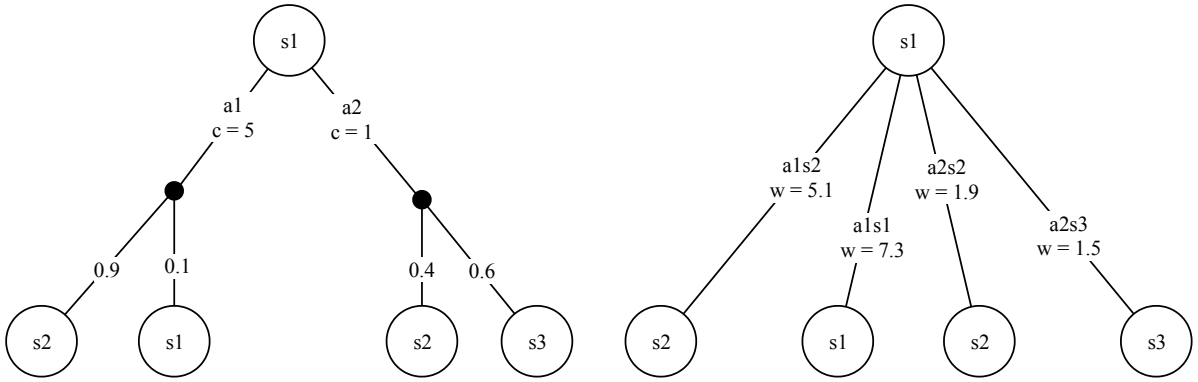
Definition 4 *An* α*-cost-likelihood* DSSPP *(*CLDSSPP*) is a* DSSPP *where*

$$W(s,a,s) = \alpha C(s,a) - log\, T(s,a,s).$$

A representation of a probabilistic search tree for an SSPP and the derived deterministic model CLDSSPP is shown in Fig. 4.3.

Finally, in order to fit the model into the regression search nature of the HPN algorithm (plan from goal to initial state), they convert a CLDSSPP into a regression cost problem as presented in definition 5.

Definition 5 *A regression cost problem (*RCP*) is a tuple* $\langle N, A, W'\rangle$ *derived from a* DSSPP $\langle S, A, W, G\rangle$*:* N *is a set of pre-images as defined below,* A *is as in the* DSSPP *and* W' *are the weights for transitions among pre-images.*



(a) Search tree of stochastic shortest path problem.

(b) Search tree for derived deterministic model (CLDSSPP), in which there is an outgoing arc for each action/outcome pair with associated weight equal to $c - \log p$

Fig. 4.3 Probabilistic search tree and its deterministic approximation

Define the weight-w pre-image of $n \in 2^S$ under action $a \in A$ to be the set of states that have a weight w arc leading to some state s' in n via action a:

$$I(n,a,w) = \{s | \exists s' \in n.\ W(s,a,s') = w\}.$$

The set N of pre-images is defined recursively starting from the goal set G of the original DSSPP*:*

- *G is an element of N;*
- *For any $n \in N$, $a \in A$, and w such that $I(n,a,w)$ is non-empty, $I(n,a,w)$ is an element of N, and $W'(I(n,a,w),a,n) = w$.*

4.3.2 MHPN: Markov HPN

The execution of the HPN algorithm extension presented before is done by monitoring the effects of actions to ensure that the action being currently selected is the first step in a plan that has a positive probability of achieving the goal. This can be translated to the condition that the current state stays in the envelope (Eq. (4.1)) of the plan which is the union of the pre-images of all the steps of the plan.

$$\text{envelope}(p) = \bigcup_{i=0}^{n-1} g_i(p) \tag{4.1}$$

The Markov HPN routine is described in Algorithm 2. It will plan, execute the plan and replan when the current state is leaving the plan envelope until the goal is reached.

Algorithm 2 Markov HPN

```
1: function MHPN(s_now, γ, α, world)
2:     p ← PLAN(s_now, γ, α)
3:     while s_now ∈ envelope(p) do
4:         i ← arg max_i s_now ∈ g_i(p)
5:         if ISPRIM(ω_i(p)) then
6:             s_now ← world.EXECUTE(ω_i(p))
7:         else
8:             s_now ← MHPN(s_now, g_i(p), NEXTLEVEL(α, ω_i(p)), world)
9:         end if
10:    end while
11:    return s_now
12: end function
13:
14: function MHPNTOP(s_now, γ, world)
15:    while s_now ∉ γ do
16:        s_now ← MHPN(s_now, γ, α_0, world)
17:    end while
18: end function
```

4.3.3 HPN in Belief Space

We now turn to the problem of sensing uncertainty. The problem of planning while having uncertainty associated to the current state of the world can be thought of as the problem of mapping current belief states (e.g. set of states satisfying "robot at position l_0 with probability greater than 0.8") into actions that drive the robot into another belief state that contains the goal state (e.g. "robot at position l_1 with probability greater than 0.95"). For that, the robot must be capable of realizing that an action such as moving its mobile base will probably scatter the probability mass around the believed localization while a sensing action will tend to concentrate the probability mass around the true state thanks to the information gathered. Planning explicitly in the belief space can enable this behavior where a convenient combination of actions (e.g., moving and sensing) are found in order to drive the belief state into the goal set.

In [33], the authors affirm that the belief state dynamics can be described as a continuous-state MDP. It is possible then to apply the same steps to convert a MDP over world states into a regression cost problem to the MDP over belief states. The result is then a regression cost problem where the nodes are sets of belief states. The planning and execution algorithm in the belief space is almost identical to the one presented before except that instead of s representing the world state we have b standing for *belief state* and one more statement

where the update of the belief is done based on the action and the observation obtained after the action. Algorithm 3 describes this modified algorithm.

Algorithm 3 HPN in belief space

```
1: function BHPN(b_now, γ, α, world)
2:     p ← PLAN(b_now, γ, α)
3:     while b_now ∈ envelope(p) do
4:         i ← argmax_i s_now ∈ g_i(p)
5:         if ISPRIM(ω_i(p)) then
6:             obs ← world.EXECUTE(ω_i(p))
7:             b_now ← b_now.UPDATE(ω_i(p), obs)
8:         else
9:             b_now ← BHPN(b_now, g_i(p), NEXTLEVEL(α, ω_i(p)), world)
10:        end if
11:    end while
12:    return b_now
13: end function
14:
15: function BHPNTOP(b_now, γ, world)
16:    while b_now ∉ γ do
17:        b_now ← BHPN(b_now, γ, α_0, world)
18:    end while
19: end function
```

4.4 Logical Characterization of Beliefs for Planning

In the approach described above, goals, when planning in the belief space, are sets of belief states. Their pre-images generated by the pre-image backchaining are also sets of belief states, and it is therefore required to have a representation of the process dynamics that modifies a belief state given an action a. The problem is then to find a way to represent belief states sets (goals and pre-images) and the belief process dynamics.

The authors of [33] use logical assertions to characterize sets of beliefs states, and symbolic operators descriptions to describe the dynamics. They do so by introducing the concept of *fluents* which is a logical predicate applied to a list of arguments. They are defined by means of a test, $\tau_f : \{\text{args}, b\} \rightarrow \{\text{true}, \text{false}\}$, where f is the fluent with arguments args, b is the belief state and f(args) is set to "hold in the belief state b" if and only if $\tau_f(\text{args}, b) = \text{true}$ (cf. Algorithm 1 line 2). Two examples for illustrating fluents definitions in a discrete-domain with n states are shown below:

- MLLoc(l): location l is the most likely location (among the n possibilities) of an object and can be defined by the following test:

$$\tau_{\text{MLLoc}}((l), b) := \forall l' \ \Pr_b(S = l) \geq \Pr_b(S = l')$$

- BLoc(l,ϵ): the object is believed to be located in location l with probability at least 1-ϵ and can be defined by the following test:

$$\tau_{\text{BLoc}}((l, \epsilon), b) := \Pr_b(S = l) \geq 1 - \epsilon$$

In [33], a generalization of these fluents for a domain with continuous quantities based on the concept of the probability near mode (PNM) of the distribution is presented.

Now that there is a compact way of representing sets of belief states, it is possible to construct symbolic operators representation, which may express the belief space dynamics. Taking as an example the action of moving an object from location l_i to l_j, [33] constructs a suitable operator description for their planning algorithm such as shown below:

MOVE(l_i, l_j)
effect: BLoc(l_j, ϵ)
choose: $l_i \in$ Locations\\{l_j}
pre: $\epsilon \geq p_{fail}$
BLoc($l_i, moveRegress(\epsilon)$)
prim: MOVEPRIMITIVE(l_i, l_j)
cost: 1

Note that effects (**effect** clause) and preconditions (**pre** clause) use one of the fluents shown before. The **choose** clause represents a generator for the initial pose l_i which is not binded to any specific value but can be chosen among possible locations excluding l_j. The **cost** here is a fixed value but could dependent on l_i and l_j.

4.5 Implementation

We developed an implementation of the approach proposed above adapted to our use case. The HPN approach requires modeling the dynamics of the world by means of fluents and operators. This process is the most challenging step when considering applying this method

for a physical system. Part of this step deals with the choice of probabilistic distributions that represent well the physical variables (such as pose in space, weigh of the object, etc.).

A secondary interesting implementation concern (at least in the particular case of our work) is the adaptation of the HPN algorithm into a non-recursive version that allows for a more decentralized and modular software architecture. We consider that it is useful for the reader to see more details about the algorithmic changes and software architecture because 1) they correspond to a meaningful part of this work 2) implementation in many cases has great bearings on the experimental side of research.

Finally, even though the hierarchical part of this approach reduces the branching factor, the pre-image backchaining process must rely on some heuristics for guiding the search of a complete plan from goal to current belief state for efficiency reasons.

4.5.1 Service-compatible, Recursion-free BHPN Algorithm

For achieving a modular, more decentralized software architecture where planning, execution, and observation can all be run inside different processes and even different computers we adapted the base BHPN algorithm. Our goal is to be able to run this algorithm as a *service* in the definition of the ROS middleware. It should, therefore, be launched by receiving a request message containing a goal and the current world state, and it should respond with the next action to be performed by the robot.

For that purpose, an iterative version of the otherwise recursive BHPN algorithm (IBHPN) was developed (see Algorithm 4). An initial call to the IBHPN routine begins the processes of planning. The procedure builds the hierarchical plans for reaching the goal and stores them in a planning stack. As soon as the first primitive operator is found, it returns the information about the action to execute to the caller (Algorithm 4 line 27). Assuming first that the caller will acts on the world (execution stage) and in turn, update its representation of it (observation stage), a second call to IBHPN will take place. Provide that the agent's goal remains the same, the procedure IBHPN evaluates the relevance of the previously computed planning stack and finds and returns the next appropriated action. As discussed before, in case b_{now} gets out of the envelope of the plan, new planning is triggered. This goes on until the agent receives a "goal achieved" return from the planner (Algorithm 4 line 5).

The proposed IBHPN fits well within the modular, decentralized software architecture needed for robust robotics. A diagram with the complete software architecture intended by our work can be seen in Fig. 4.4. The organization is a very classical way of representing the different layers of planning for autonomous robots. We highlight on this diagram where the algorithms presented in the thesis would be integrated. Besides, all the code necessary

Algorithm 4 Iterative BHPN

$p_{\text{stack}} \leftarrow \varnothing$
$\gamma_{\text{final}} \leftarrow \varnothing$
function IBHPN(b_{now}, γ)
  if γ holds in b_{now} **then**
    return {GoalAchieved, $\varnothing$}
  end if
  while true **do**
    if p_{stack} is empty **then**
      $\gamma_{\text{final}} \leftarrow \gamma$
      $p \leftarrow$PLAN($b_{\text{now}}, \gamma, 0$)
      if p is empty **then**
        return {Fail, $\varnothing$}
      end if
      p_{stack}.PUSHBACK($\{p, \gamma, 0\}$)
    end if
    if γ is not equal to γ_{final} **then**
      return {Fail, $\varnothing$}
    end if
    while p_{stack} is not empty **do**
      $p, \gamma_{\text{inter}}, \alpha \leftarrow p_{\text{stack}}$.POPBACK()
      if p is empty or γ_{inter} holds in b_{now} or $b_{now} \notin$ENVELOPE(p) **then**
        continue
      end if
      $i \leftarrow \arg\max_i b_{\text{now}} \in g_i(p)$
      if ISPRIM($\omega_i(p)$) **then**
        p_{stack}.PUSHBACK($\{p, \gamma_{\text{inter}}, \alpha\}$)
        return {ValidActionFound, $\omega_i(p)$}
      else
        p_{stack}.PUSHBACK($\{p, \gamma_{\text{inter}}, \alpha\}$)
        $\gamma_{\text{new}} \leftarrow$NEXTGOAL($i, p, \gamma_{\text{inter}}$)
        $\alpha_{\text{new}} \leftarrow$NEXTLEVEL($\alpha, \omega_i(p)$)
        $p \leftarrow$PLAN($b_{\text{now}}, \gamma_{\text{new}}, \alpha_{\text{new}}$)
        p_{stack}.PUSHBACK($\{p, \gamma_{\text{new}}, \alpha_{\text{new}}\}$)
      end if
    end while
  end while
end function
function NEXTGOAL($i, p, \gamma_{\text{inter}}$)
  if $i + 1 \geq p$.LENGTH() **then**
    return γ_{inter}
  else
    return $g_{i+1}(p)$
  end if
end function

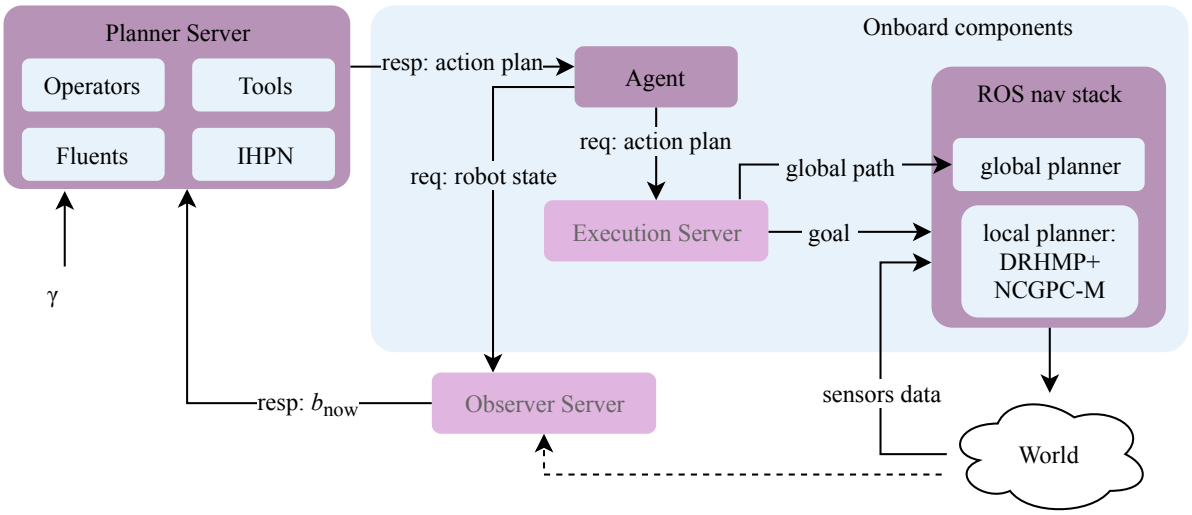


Fig. 4.4 Complete software architecture for planning and actions execution on a mobile robot using ROS.

for interfacing the different components was implemented. A connection to Gazebo for simulating the "world" part of the diagram was made for initial testing and experimenting.

Unfortunately, due to lack of time, these integration tests were not completed during the time of this research. The missing components are shown in the diagram by blocks with fading colors (*Execution server* and *Observer server*). Nevertheless, the components that were developed enabled us to start some simple planning experiments presented in the remainder of this chapter by mocking some components, namely the world, execution, and observation parts.

4.5.2 Probability Distributions Underlying the Belief States

For representing probability distribution of poses two different distributions were used: for positions, x and y were represented as two independent random variables distributed normally $\left(X \sim \mathcal{N}\left(\mu, \sigma^2\right)\right)$; for orientation we used the von Mises distribution (also called wrapped normal distribution) $\left(\Psi \sim \text{vMises}\left(\mu, \kappa\right)\right)$.

Note

The von Mises distribution can be represented as follows:

$$f(x|\mu,\kappa) = \frac{e^{\kappa\cos(x-\mu)}}{2\pi I_0(\kappa)} \tag{4.2}$$

with $I_0(\kappa)$ being the modified Bessel function of order 0 [3]. A graphical representation of the distribution can be seeing in Figure 4.5. The parameters μ and $1/\kappa$ are analogous to μ and σ^2 (the mode and variance) in the normal distribution.

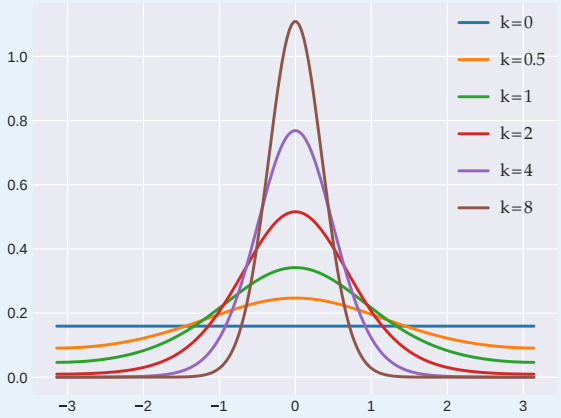


Fig. 4.5 Probability density function of von Mises distribution for different κ. Support is $[-\pi, \pi]$ and $\mu = 0$

4.5.3 Fluents

Seven different fluents were devised in order to solve simple task planning problems representative of logistics scenario where a number of objects have to be moved in order to reach a goal configuration. We added support for the notion that some entities in this simplified world can be controlled (moved between locations and ordered to pick another entity).

In the definitions detailed below, we used several functions for querying values from the belief state ($\sigma_{*,*}(b)$, $\mu_{*,*}(b)$, $\text{region}_{*,*}(b)$, $\text{held}_*(b)$, $\text{pickable}_*(b)$, $\text{controllable}_*(b)$). We assume that whatever the belief representation may be, it can provide this information about objects in the world.

The seven fluents are briefly explained below:

- PSWD(A,δ,ϵ)

  PSWD stands for "Precisely Somewhere Within Delta". This logical predicate models precision about random variables defining a pose in space (it does not model accuracy). Its test evaluates to true if the probability mass of a random variable within δ from its mode is greater or equal to $(1-\epsilon)$. In other words, the smaller the ϵ, the more precise the current belief about the variable's value has to be in order for the testing of PSWD to be verified true.

For a 2D pose of an entity A, we can apply this idea to three independent random variables representing a pose and the PSWD fluent can be defined as follows:

$$\begin{aligned}\tau_{\text{PSWD}}((A,\delta,\epsilon),b) \quad := \quad & \text{PNVN}\left(\delta_x,\sigma_{x,A}(b),0\right) \geq 1-\epsilon_x \wedge \\ & \text{PNVN}\left(\delta_y,\sigma_{y,A}(b),0\right) \geq 1-\epsilon_y \wedge \\ & \text{PNVVM}\left(\delta_\psi,\sigma_{\psi,A}(b),0\right) \geq 1-\epsilon_\psi\end{aligned}$$

where the wedge symbol $\wedge$ represents the logical operator *and* and PNVN (or "Probability Near Value for Normal distribution") and PNVVM (or "Probability Near Value for Von Mises distribution") are defined as follows

$$\begin{aligned}\text{PNVN}(\delta,\sigma,\mu) \quad &= P\left(\mu-\delta < X \leq \mu+\delta\right) = \Phi_X(\mu+\delta) - \Phi_X(\mu-\delta) \mid X \sim \mathcal{N}\left(\mu,\sigma^2\right), \\ \text{PNVVM}(\delta,\sigma,\mu) \quad &= P\left(\mu-\delta < \Psi \leq \mu+\delta\right) = \Phi_\Psi(\mu+\delta) - \Phi_\Psi(\mu-\delta) \mid \Psi \sim \text{vMises}\left(\mu,\kappa\right)\end{aligned}$$

with Φ being the cumulative distribution function (CDF) of variables X and Ψ.

- TWD(A,δ,q)

  TWD (or "There Within Delta") models knowledge about accuracy (not precision). Its test evaluates to true if the realization of a random variable is within δ from its believed mode q. For the 2D case we can write:

$$\begin{aligned}\tau_{\text{TWD}}((A,\delta,q),b) \quad := \quad & \|q_x - \mu_{x,A}(b)\| \leq \delta_x \wedge \\ & \|q_y - \mu_{x,A}(b)\| \leq \delta_y \wedge \\ & \|q_\psi - \mu_{\psi,A}(b)\| \leq \delta_\psi\end{aligned}$$

- RPC$(R,\{A,B,...\},\epsilon)$

  RPC (or "Region Probably Clear") models knowledge of a region being cleared (with given certainty) of all entities (represented by the set $\mathscr{E}$) except the ones explicitly specified:

$$\tau_{\text{RPC}}((R,\{A,B,...\},\epsilon),b) \quad := \quad R \bigcap \text{region}_{E,\epsilon}(b) = \varnothing \quad \forall E \in \mathscr{E} \backslash \{A,B,...\}$$

- PICKED(A)

  It models knowledge about an entity A (e.g. object) being held by any other entity (e.g. robotic arm):

$$\tau_{\text{PICKED}}((A),b) := \text{held}_A(b)$$

- HOLDS(A, B)

  It models knowledge about an entity A (e.g. robotic arm) holding another entity B (e.g. object):

$$\begin{aligned}\tau_{\text{HOLDS}}((A,B),b) \quad := \quad & \text{held}_B(b) \wedge \\ & \text{region}_{B,1}(b) \bigcap \text{region}_{A,1}(b) = \text{region}_{B,1}(b)\end{aligned}$$

  The information that the entity B is held by A in particular and not some other entity is conveyed by testing if the region occupied by B is encompassed by A's region.

- PICKABLE(A)

  It models knowledge about an entity being "pickable" by another entity:

$$\tau_{\text{PICKABLE}}((A),b) := \text{pickable}_A(b)$$

- CONTROLLABLE(A)

  It models the property of being an agent/robot (controllable entities):

$$\tau_{\text{CONTROLLABLE}}((A),b) := \text{controllable}_A(b)$$

Fluents also have intrinsic methods for defining if they entail or contradict other fluents. These methods are fundamental for generating pre-images from operators during the process of growing the search tree.

Another important observation is that we allowed ourselves to mix fluents that model uncertainty as well as fluents that do not. This was done in order to simplify how transitions after some actions would occur, but a more complete solution would rewrite some of those fluents.

4.5.4 Operators

Operators are, generally speaking, actions that can be performed by the planning agent. Some are very close to actual physical actions, and others are more logical actions meant to translate one pre-image into another (as if they were reasoning actions such as inference/deduction).

An operator is defined by its pre-requirements, effects, generators, primitive operation and cost. We rely on several auxiliary functions for defining these fluents: swept(A, p) defines

the area swept by the object A while moving along the path p, PICKINGPOSEFOR(l) returns a position from which a robot can pick an object at location l.

Operators used in this work are described below:

- MOVE($A, q_i, q_f, \delta, \epsilon$): this operator will move an object A from position q_i to position q_f

  effect: TWD(A, δ, q_f) $\wedge$ PSWD(A, δ, ϵ)

  choose: $q_i \in$ Locations$\backslash\{q_f\}$

  $p \in$ paths(A, q_i, q_f, g_i, b)

  pre: RPC$\big($swept$(A, p), \{A\}, \epsilon_r\big)$ $\wedge$ TWD$\big(A, \delta, q_f\big)$ $\wedge$

  PSWD(A, δ, ϵ_r) $\wedge$ CONTROLLABLE(A)

  prim: MOVEPRIMITIVE($A, q_i, q_f, \delta, \epsilon$)

  cost $\propto$ length(p)

- CLEAR($R, \mathscr{P}, \epsilon$): this operator will clear a region R from any entity that do not belong to the set of entities $\mathscr{P}$

  effect: RPC($R, \mathscr{P}, \epsilon$)

  choose: $q_E \in$ Locations $\forall\ E \in \mathscr{E}\backslash\mathscr{P} \mid \text{region}_{E,\epsilon}(q_E) \bigcap R = \varnothing$

  pre: TWD$\big(E, \delta, q_E\big)$ $\wedge$ PSWD(E, δ, ϵ_r) $\forall\ E \in \mathscr{E}\backslash\mathscr{P}$

  prim: CLEARPRIMITIVE($R, \mathscr{P}, \epsilon$)

  cost: 0

- PICK(A, B, δ, ϵ): this operator will make entity A pick entity B

  effect: PICKED(B) $\wedge$ HOLDS(A, B) $\wedge$ PSWD(B, δ, ϵ) $\wedge$ TWD(B, δ, q_A)

  choose: $q_i \leftarrow$PICKINGPOSEFOR($\mu_B(b)$)

  pre: PSWD(A, δ, ϵ) $\wedge$ TWD(A, δ, q_i)

  prim: PICKPRIMITIVE(A, B)

  cost $\propto$ $A_{\text{dim}} + B_{\text{dim}}$

- PLACE($A, B, q_f, \delta, \epsilon$): this operator will make entity A place entity B at position q_f

  effect: TWD(B, δ, q_f) $\wedge$ PSWD(B, δ, ϵ) $\wedge$ $\neg$PICKED(B)

  choose: $E \in$ Locations$\backslash\{l_j\}$

  $q_i \leftarrow$PICKINGPOSEFOR(q_f)

pre: PICKED(B) $\wedge$ PICKABLE(B) $\wedge$ RPC(region$_{B,\epsilon}(q_f), B, \epsilon$) $\wedge$
PSWD(A, δ, ϵ) $\wedge$ TWD(A, q_i, δ) $\wedge$ HOLDS(A, B)

prim: PLACEPRIMITIVE(A, B)

cost $\propto A_{\text{dim}} + B_{\text{dim}}$

A_{dim} is the characteristic dimension of the entity, notably the diameter of the circle containing the region$_{A,1}$. The symbol $\neg$ represents the logical operator *not*.

4.6 Preliminary Experimental Results

With the implementation of the IBHPN, Agent, Tools, Fluents, and Operators represented in Fig. 4.4 and discussed before, some planning tests were possible using simplified use-cases. At first, a complete symbolic example case was studied were few operators (around 4) existed and were able to cause transitions from states represented by single letters. Then a second more interesting scenario where simple 2D geometric rules exist (loosely inspired by tetris-like games).

4.6.1 Planning in a Simplified 2D World

For this experiment, we simplified the set of fluents and actions. Only Move and Clear are used, and all entities can be controlled (if they can move).

The goal for this example can be defined simply by 2 two fluents:

$$\gamma = \begin{cases} \text{PSWD}(A, \delta_A, \epsilon_A) \wedge \\ \text{TWD}(A, \delta_A, q = [0.14, 0.14, 0.0]) \end{cases}$$

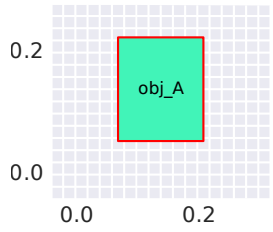


Fig. 4.6 Representation of goal belief state γ

Constants δ_A and ϵ_A where chosen to be equals to $[0.05, 0.05, 0.05]$ both.

The initial belief state was defined as the set of all entities, their attributes, and the current value of those attributes. A given entity had the following list of attributes in its representation: mode of its pose in 2D space (q), standard deviation of that pose as if each dimension was an independent random variable (σ), type of footprint (or type of shape), footprint values, color and lastly the movable attribute. An image summarizing the

description of the initial belief state bs_0 can be seen in Fig. 4.7a. Object tagged Immov was the only immovable one. In order to simplify the plan, we set the initial uncertainties to very low values (0.0003).

4.6.2 Planning in a Simplified 2D World with a Robot

In this second experiment, all defined fluents and operators were available. The objective was to have a single robot able to move objects, some unmovable objects, and a single object to be moved from initial to final position. As before, no execution error was allowed, thus no replanning was triggered yielding a single initial plan that is perfectly executed. Again, since all operations are primitive actions, no abstraction level besides the first (zero) is exploited.

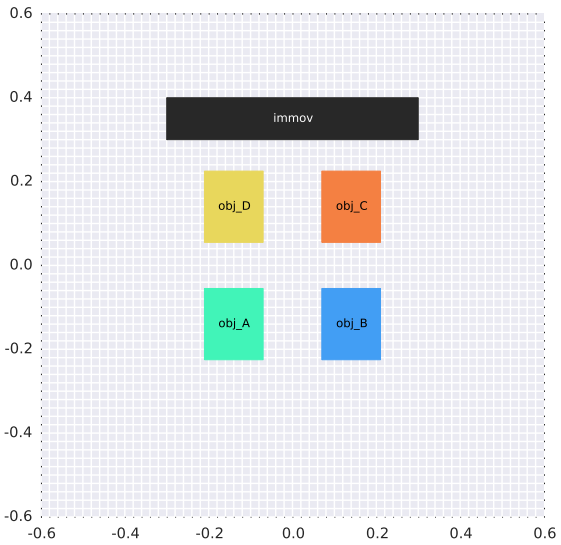
Fig. 4.10 shows the resulting sequence of actions for archiving the goal of having object A at its goal location. Fig. 4.11 and 4.12 help to illustrate the obtained plan. For more details about the plans, refer to Appendix C.

4.7 Conclusions

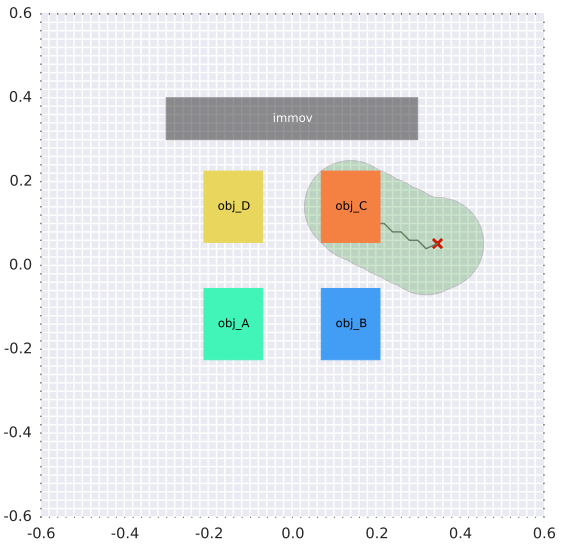
The benefits of close integration of logical and geometric planning has long been recognized [61]. It is a crucial element for building truly autonomous robots that can execute high-level tasks in real environments.

The work done on this subject in this thesis was very preliminary. The HPN approach was shown to have some interesting properties, and a recursion-free implementation was realized and applied to simplified use cases. However, the most useful capabilities, namely its hierarchical structure, were barely exploited in the performed experiments.

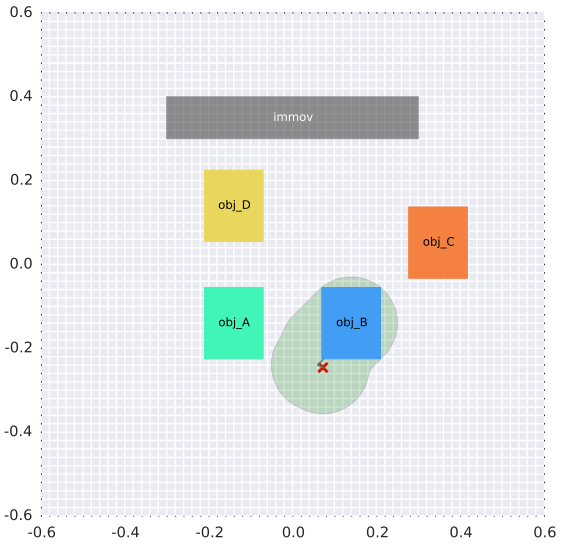
Further work in this subject should exploit postponing preconditions of actions to create abstraction levels for hierarchical planning, and better models for modifying the belief states upon actions.



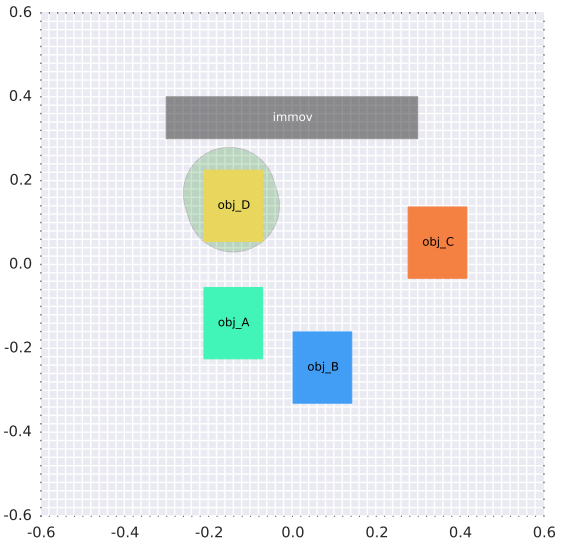
(a) Initial state corresponding to $\mathbf{bs_0}$



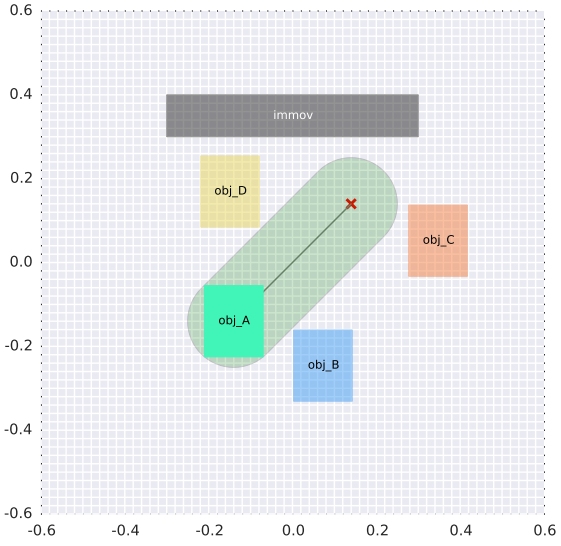
(b) Pre-image g_8 and its associated operator



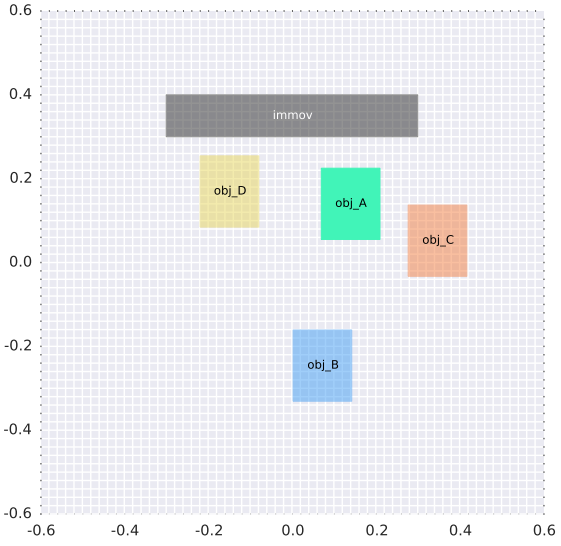
(c) Pre-image g_7 and its associated operator



(d) Pre-image g_5 and its associated operator



(e) Pre-image g_1 and its associated operator



(f) Final configuration when goal is reached

Fig. 4.7 First geometric case

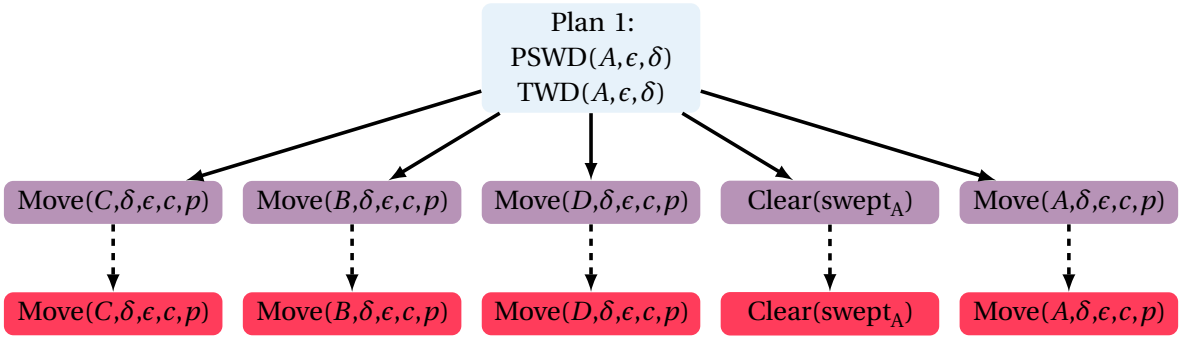


Fig. 4.8 Planning and execution for the first geometric case experiment

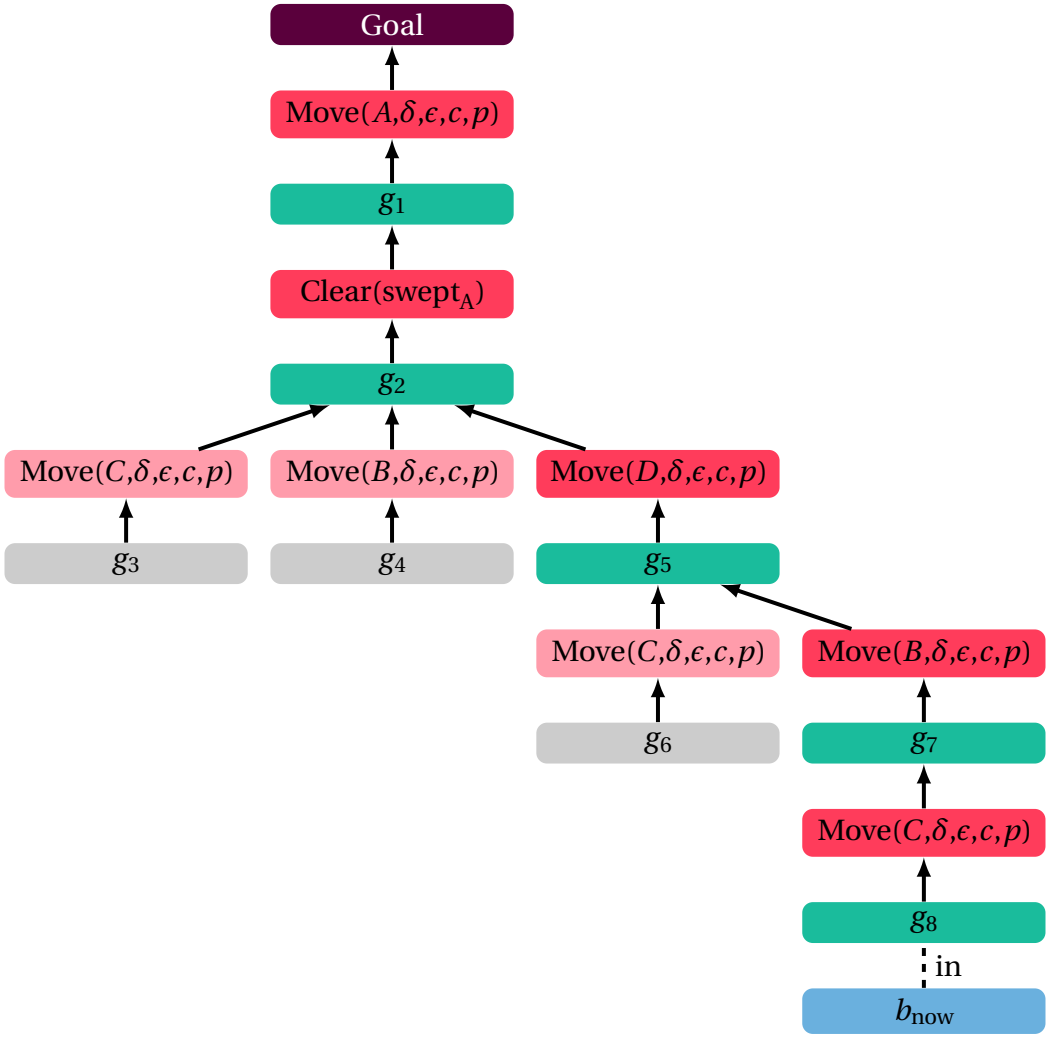
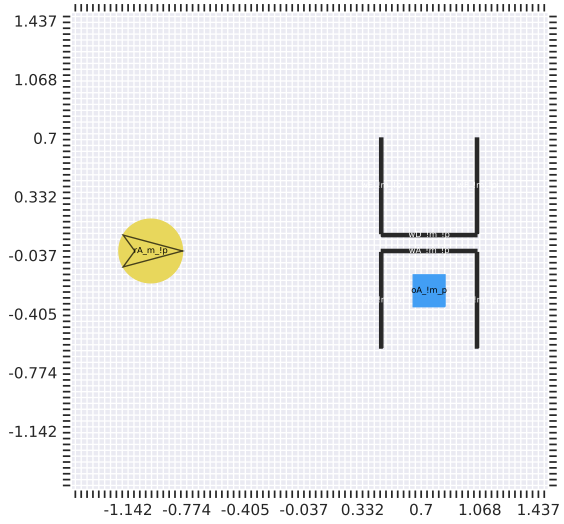
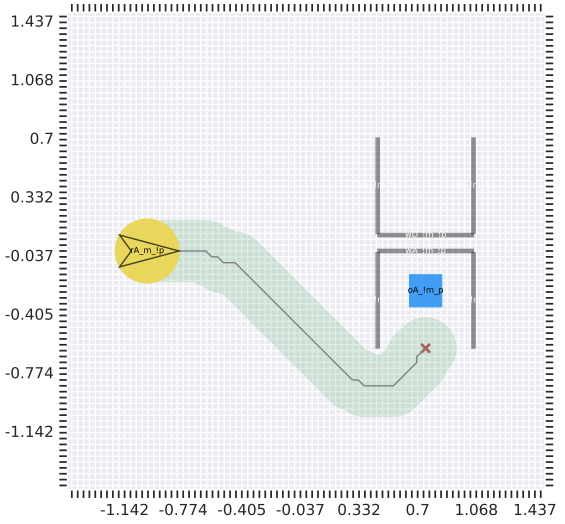


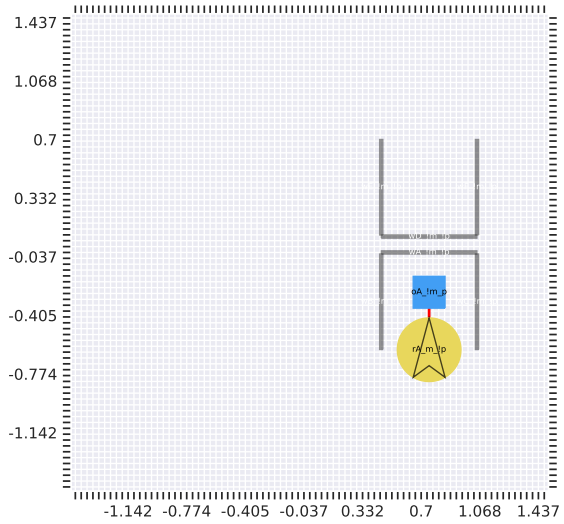
Fig. 4.9 Plan 1 for the first geometric case showing pre-images



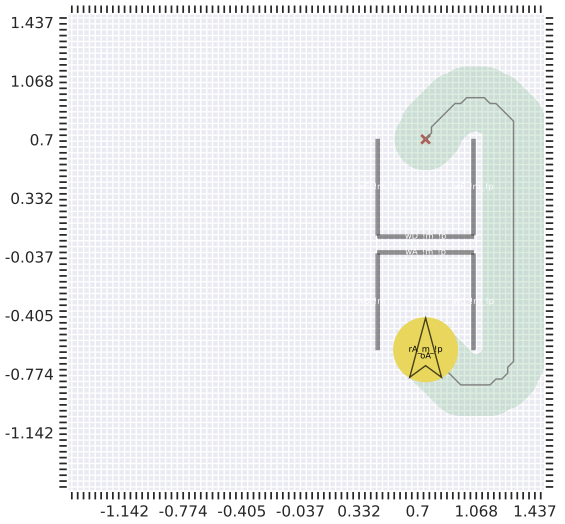
(a) Initial state corresponding to $\mathbf{bs_0}$



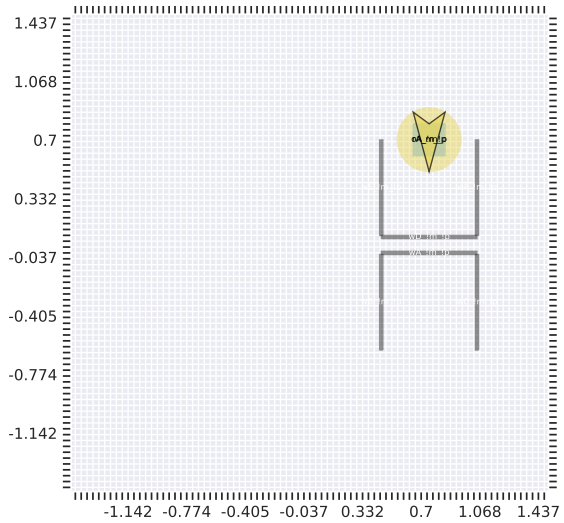
(b) Pre-image g_8 and its associated operator



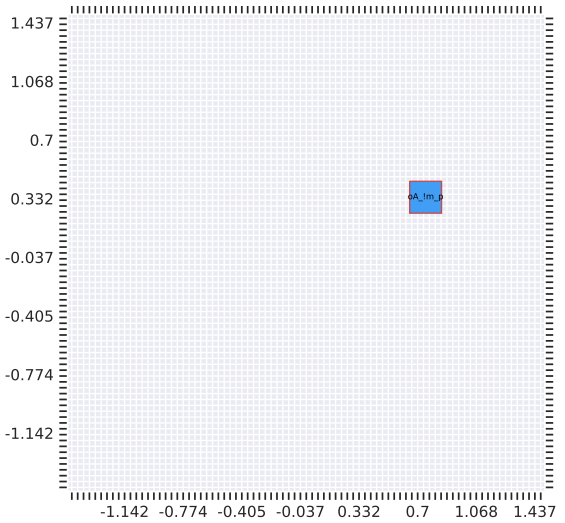
(c) Pre-image g_7 and its associated operator



(d) Pre-image g_5 and its associated operator



(e) Pre-image g_1 and its associated operator



(f) Final configuration when goal is reached

Fig. 4.10 Second geometric case

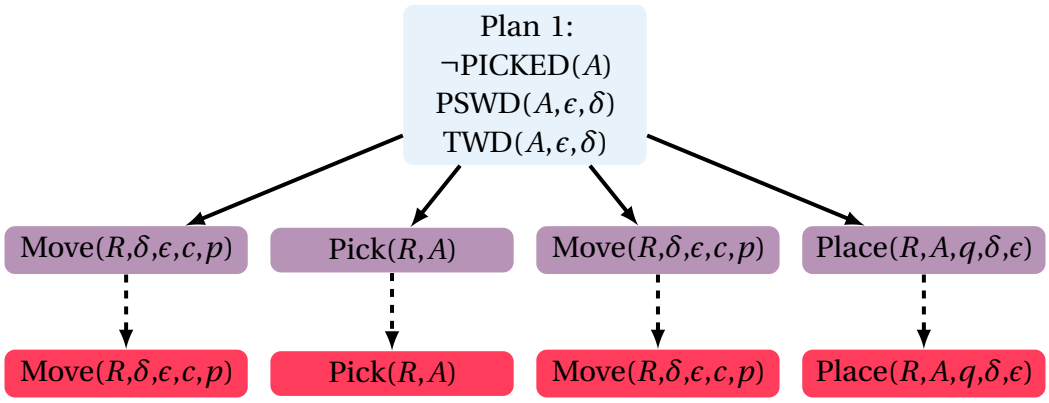


Fig. 4.11 Planning and execution for the second geometric case experiment

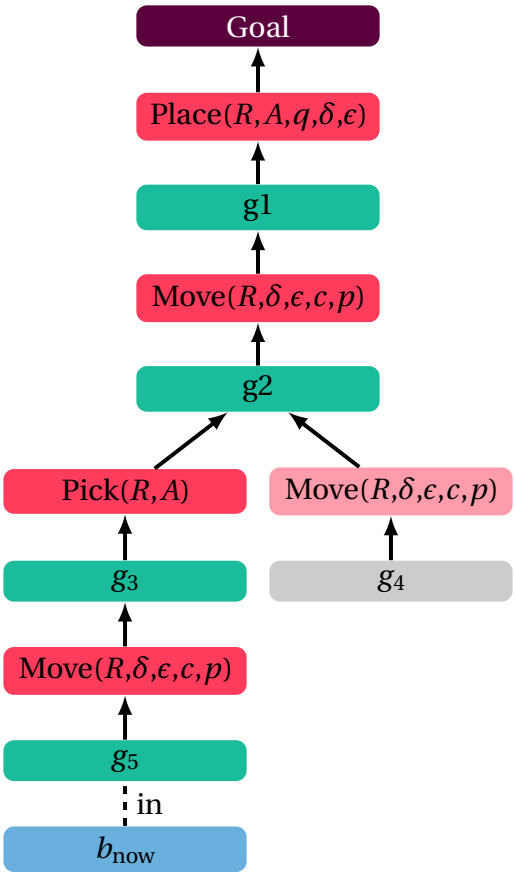


Fig. 4.12 Plan 1 for the second geometric case showing pre-images

Chapter 5

General Conclusions and Perspectives

5.1 Summary and Discussions

Throughout the previous chapters, the subject of planning for a multi-robot system was discussed. The main points exploited were threefold: a local distributed multi-robot trajectory planning, a feedback model predictive controller, and (albeit in a more superficial manner) the integration of a hierarchical task planner with the rest of the work.

Our approach to trajectory planning consisted of an online, distributed, multi-robot algorithm based on mathematical programming and receding horizon techniques called DRHMP. The distributed coordination of the robots was achieved by a two-step process where first trajectories are generated by ignoring coupling constraints between robots, and second, by exchanging those first solutions, the robots finally incorporate the coupling constraints to find the final, non-conflicting trajectories.

Given that the DRHMP solutions cover a future time horizon T_p from the start of each planning stage, we proposed a new Nonlinear Continuous Generalized Predictive Control (NCGPC-M) to take full advantage of that information. The final control law equations are based on a direct dynamic model of unicycle-like vehicles. NCGPC-M was capable of improving the quality of the trajectory tracking of the mobile robots compared to two other control laws, especially in the presence of high dynamics (meaningful inertial mass and accelerations).

The third part of this work, studied an approach called HPN (Hierarchical Planning in the Now) for task planning. In this approach planning is performed in a mixed logical and geometric belief space allowing for robustness against uncertainty. Its integration with the rest of the work was roughly theorized in a schematics presented in Fig. 4.4. The HPN algorithm, which is naturally expressed in a recursive form, was rewritten in an iterative one, adapted for the integration with the other components of this work. A few domain-specific

actions used as input by the HPN were designed to support our use-case of mobile wheeled vehicles.

In order to evaluate those approaches, we have performed a few experiments. We have shown results of the DRHMP operating in simulation with three robots, and in real world with two vehicles. In order to the optimizer underlying the DRHMP approach to converge to locally optimal, kinematically feasible, collision-free trajectories the period of the planner (given by T_c) was set to 0.3 s (while running on an Intel Xeon CPU 2.53GHz processor).

When testing the NCGPC-M controller, in those same real world conditions, it presented an RMS position tracking error of 0.44 cm and angular tracking error of 0.34° (compared to the 1.17 cm and 0.75° with the previous existing NCGPC).

Some of the strong hypotheses made by this work are listed below:

- nonholonomic unicycle-like robots;
- circular footprint;
- known direct dynamic model of the vehicles;
- communication delays many times smaller than the computation time window T_c;
- environment dynamics (excluding the robots) that can be considered static during T_c;
- stable numeric differentiation of constraints (equations and inequations) and objective function;

5.2 Perspectives

Our work leads to several improvement ideas summarized below.

5.2.1 Avoiding Numerical Differentiation

Many optimizers, such as the SLSQP algorithm used for optimizing trajectories in Chapter 2, are gradient-based, meaning they require the derivatives of objective functions and constraints to find a solution. Throughout our work, we used numerical differentiation to approximate gradients whenever it was needed. This approach has the advantage of being generic and fast to compute, however, round-off errors are inherent to it, and those errors may represent a problem to the optimizers. Indeed, we observed some instabilities of the SLSQP algorithm that may correlate to the round-off errors from numerical differentiation.

A possible alternative to that approach is the use of either *symbolic differentiation* [81] or *automatic differentiation* (AD) [9]. We attempted to use symbolic differentiation software (Matlab) for finding the exact expressions for the Jacobian and Hessian matrices[1]. For at least

[1] matrices formed from the first and second-order partial derivatives, respectively

part of the constraints, that approach was successful, but the complexity of the functions was high enough to make them useless as the running time grew considerably.

Automatic differentiation remains to be tested. It could potentially solve the problem of round-off errors and inefficiency present on the two other approaches. If C++17 is the programming language being used, a prominent library for automatic differentiation can be found in [46]. Many other libraries for virtually all programming languages exist; the website in [14] lists 67 AD tools for 21 different languages in addition to a publication database on AD of 1528 items.

5.2.2 Conversion from Sensor Data to Geometric Objects

The local interpolation of the occupancy grid for generating continuously differentiable equations for obstacle avoidance, although used with success in real experiments, has its limitations. The approach does not generalize well for complex environments with non-convex obstacles and robot's footprints with aspect ratio far from one.

Work presented in [23, 65] on *Density-based spatial clustering of applications with noise* (DBSCAN) and *Random sample consensus* (RANSAC) can be used to convert occupancy grid information or even point cloud data from lasers or cameras into a geometric description of the environment. This approach might be a viable alternative to describe the environment and accomplish obstacle avoidance in our planning approach. Fig. 5.1 shows an image from an existing project that uses the mentioned techniques to accomplish the generation of convex polygons from occupancy grids.

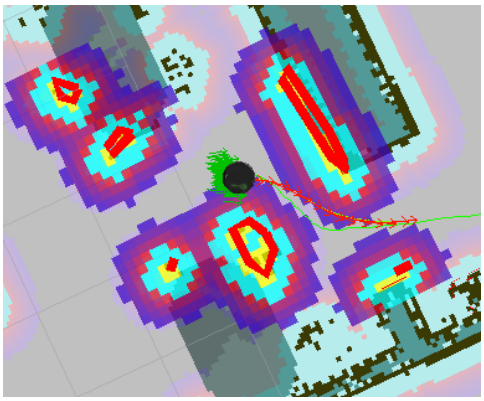


Fig. 5.1 Example of costmap conversion to convex polygons (red edges close to yellow cells)
Source: ROS Wiki page on costmap converter (http://wiki.ros.org/costmap_converter)

5.2.3 Alternative Optimization Solver

An alternative to the use of SLSQP optimizer is the CFSQP [85] (Contraint Feasible Sequential Quadratic Programming) algorithm used, for instance, in work presented in [20]. CFSQP approach guarantees that every iteration of the optimization process generates a set of control variables that respects the constraints.

The advantage of such an alternative over the SLSQP in our DRHMP approach is that, in cases where T_c (computation time reserved for finding a solution) elapses before the convergence of the optimizer, the solution from the last iteration could still be safely used, since it is guaranteed to respect the NLP constraints. For more information about optimization solvers refer to Appendix A, section A.1.

5.2.4 Integrating Task and Motion Planning

Regarding the preliminary work done on the HPN approach, the exploration of hierarchical knowledge about the actions presented in Chapter 4 would be the natural course of development, followed by a better understating and formulation of the probability distributions regressions under those actions.

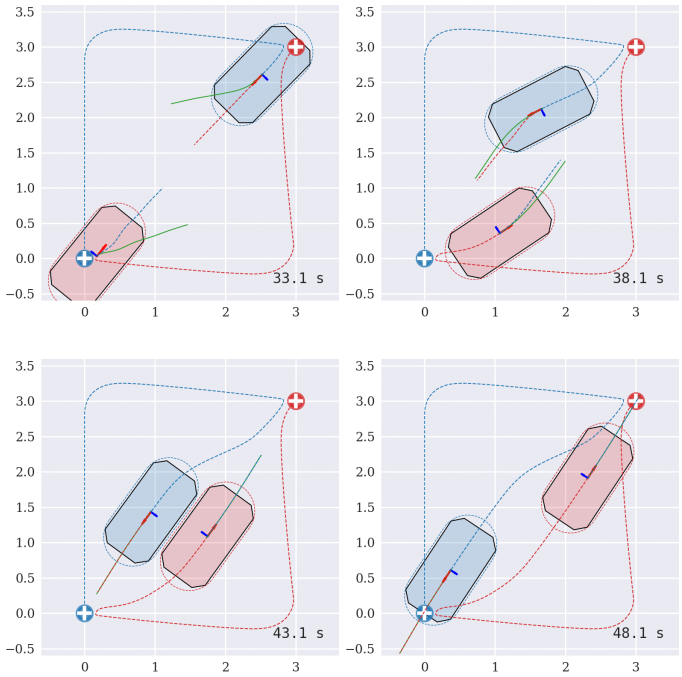


Fig. 5.2 Non-circular footrprint example

5.2.5 Future Work

Follow up work on these subjects, especially the DRHMP trajectory planning approach, and the NCGPC-M controller, are being conducted both at CEA Sacaly under Eric Lucet's supervision and at iFollow SAS (a startup robotics company based in Paris region where the author of this thesis is currently working).

In particular, a generalization of the DRHMP regarding the robot's footprint was tested by using two different levels of footprints; first a high fidelity level using a n-sides polygon and second an approximated level using conjunction of simpler geometric primitives (circles and segments of lines). Fig. 5.2 gives an overview of a planning test similar to the one reprensented in Fig. 2.22 for a polygon footrprint of 8 sides with aspect ratio close to 1.8.

References

[1] Abichandani, P., Benson, H., Kam, M., et al. (2013). Mathematical programming approaches for multi-vehicle motion planning: Linear, nonlinear, and mixed integer programming. *Foundations and Trends® in Robotics*, 2(4):261–338.

[2] Abichandani, P., Ford, G., Benson, H. Y., and Kam, M. (2012). Mathematical programming for multi-vehicle motion planning problems. In *2012 IEEE International Conference on Robotics and Automation*, pages 3315–3322.

[3] Abramowitz, M. and Stegun, I. A. (1972). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. National Bureau of Standards Applied Mathematics Series 55. Tenth Printing.* ERIC.

[4] Alicja, M. (1999). A new universal adaptive tracking control law for nonholonomic wheeled mobile robots moving in r3 space. In *IEEE International Conference on Robotics and Automation*.

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

[6] Arney, T. (2007). An efficient solution to autonomous path planning by approximate cell decomposition. In *2007 Third International Conference on Information and Automation for Sustainability*, pages 88–93. IEEE.

[7] Asano, T., Asano, T., Guibas, L., Hershberger, J., and Imai, H. (1986). Visibility of disjoint polygons. *Algorithmica*, 1(1-4):49–63.

[8] Ashoorirad, M., Barzamini, R., Afshar, A., and Jouzdani, J. (2006). Model Reference Adaptive Path Following for Wheeled Mobile Robots. In *2006 International Conference on Information and Automation*, pages 289–294.

[9] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2018). Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153).

[10] Bde Boor, C. (2001). A practical guide to splines, revised edition.

[11] Bell, T., O'Connor, M., Jones, V., Rekow, A., Elkaim, G., and Parkinson, B. (1998). Realistic autofarming closed-loop tractor control over irregular paths using kinematic gps. *The Journal of Navigation*, 51(3):327–335.

[12] Borrelli, F., Subramanian, D., a.U. Raghunathan, and Biegler, L. (2006). MILP and NLP Techniques for centralized trajectory planning of multiple unmanned air vehicles. *2006 American Control Conference*, pages 5763–5768.

[13] Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122.

[14] Bücker, M. (2000). autodiff.org. http://www.autodiff.org/?module=Tools&language=ALL. Accessed: 2019-02-11.

[15] Canny, J. and Reif, J. (1987). New lower bound techniques for robot motion planning problems. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 49–60. IEEE.

[16] Carsten, J., Rankin, A., Ferguson, D., and Stentz, A. (2007). Global path planning on board the mars exploration rovers. In *2007 IEEE Aerospace Conference*, pages 1–11. IEEE.

[17] Choset, H. M., Hutchinson, S., Lynch, K. M., Kantor, G., Burgard, W., Kavraki, L. E., and Thrun, S. (2005). *Principles of robot motion: theory, algorithms, and implementation*. MIT press.

[18] Curnow, R. and Lichvar, M. (2019). chrony utility. https://chrony.tuxfamily.org. Accessed: 2019-10-03.

[19] De La Cruz, C. and Carelli, R. (2006). Dynamic modeling and centralized formation control of mobile robots. *IECON 2006-32nd Annual Conference on IEEE Industrial Electronics*, pages 3880–3885.

[20] Defoort, M., Kokosy, A., Floquet, T., Perruquetti, W., and Palos, J. (2009). Motion planning for cooperative unicycle-type mobile robots with limited sensing ranges: A distributed receding horizon approach. *Robotics and Autonomous Systems*, 57(11):1094–1106.

[21] developpers, S. (2019). Scipy - scientific computing tools for python. http://www.scipy.org/. Accessed: 2019-10-03.

[22] Douthwaite, J. A., Zhao, S., and Mihaylova, L. S. (2019). Velocity obstacle approaches for multi-agent collision avoidance. *Unmanned Systems*, 7(01):55–64.

[23] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.

[24] Filotheou, A., Nikou, A., and Dimarogonas, D. V. (2018). Robust decentralized navigation of multi-agent systems with collision avoidance and connectivity maintenance using model predictive controllers. *arXiv preprint arXiv:1804.09039*.

[25] Fiorini, P. and Shiller, Z. (1998). Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772.

[26] Fox, D., Burgard, W., and Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33.

[27] Gartner (2019). Gartner predicts 2019 for supply chain operations. https://www.gartner.com/smarterwithgartner/gartner-predicts-2019-for-supply-chain-operations. Accessed: 2019-10-03.

[28] Gayle, R., Sud, A., Lin, M. C., and Manocha, D. (2007). Reactive deformation roadmaps: motion planning of multiple robots in dynamic environments. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3777–3783. IEEE.

[29] Gizmag (2014). Autonomous robots are helping to pack your amazon orders. http://www.gizmag.com/amazon-kiva-fulfillment-system/34999/. Accessed: 2019-10-03.

[30] Hamerlain, F., Achour, K., Floquet, T., and Perruquetti, W. (2007). Trajectory tracking of a car-like robot using second order sliding mode control. In *2007 European Control Conference (ECC)*, pages 4932–4936.

[31] Hopcroft, J., Schwartz, J., and Sharir, M. (1984). On the complexity of motion planning for multiple independent objects; pspace- hardness of the "warehouseman's problem". *The International Journal of Robotics Research*, 3(4):76–88.

[32] Hoy, M., Matveev, A. S., and Savkin, A. V. (2015). Algorithms for collision-free navigation of mobile robots in complex cluttered environments: A survey. *Robotica*, 33(3):463–497.

[33] Kaelbling, L. P. and Lozano-Perez, T. (2013). Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32:1194–1227.

[34] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580.

[35] Keil, J. M. and Sack, J.-R. (1985). Minimum decompositions of polygonal objects. In *Machine Intelligence and Pattern Recognition*, volume 2, pages 197–216. Elsevier.

[36] Kelly, M. P. (2017). Transcription methods for trajectory optimization: a beginners tutorial. *arXiv preprint arXiv:1707.00284*.

[37] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous Robot Vehicles*, pages 396–404. Springer Science and Business Media.

[38] Kraft, D. (1988). *A software package for sequential quadratic programming*. DLR German Aerospace Center – Institute for Flight Mechanics, Koln, Germany.

[39] Krid, M., Benamar, F., and Lenain, R. (2016). A new explicit dynamic path tracking controller using Generalized Predictive Control. *International Journal of Control, Automation and Systems*, pages 1–10.

[40] Kurniawati, H., Hsu, D., and Lee, W. S. (2008). Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, volume 2008. Zurich, Switzerland.

[41] La Salle, J. and Lefschetz, S. (2012). *Stability by Liapunov's Direct Method with Applications by Joseph L Salle and Solomon Lefschetz*, volume 4. Elsevier.

[42] Lages, W. F. and Hemerly, E. M. (1998). Adaptive Linearizing Control of Mobile Robots. *IFAC Proceedings Volumes*, 31(31):23–28.

[43] Latombe, J.-C. (1991). *Robot Motion Planning*, volume 124. Springer US, Boston, MA.

[44] Laumond, J.-P. P. (1998). *Robot Motion Planning and Control*. Springer-Verlag, Berlin, Heidelberg.

[45] LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

[46] Leal, A. (2018). autodiff - Automatic Differentiation in C++. https://autodiff.github.io. Accessed: 2019-02-11.

[47] Levihn, M., Scholz, J., and Stilman, M. (2013). Hierarchical decision theoretic planning for navigation among movable obstacles. In *Algorithmic Foundations of Robotics X*, pages 19–35. Springer.

[48] Lu, W. C., Duan, L., Fei-Bin, H., and Mora-Camino, F. (2008). Differential flatness applied to vehicle trajectory tracking. In *2008 27th Chinese Control Conference*, pages 242–247. IEEE.

[49] Maestre, J. M., Negenborn, R. R., et al. (2014). *Distributed model predictive control made easy*, volume 69. Springer.

[50] Marino, R. (1997). Adaptive control of nonlinear systems: Basic results and applications. *Annual Reviews in Control*, 21:55–66.

[51] Marino, R. and Tomei, P. (1995). *Nonlinear Control Design: Geometric, Adaptive, and Robust*. Prentice Hall. Google-Books-ID: HQprQgAACAAJ.

[52] Mehrez, M. W., Sprodowski, T., Worthmann, K., Mann, G. K., Gosine, R. G., Sagawa, J. K., and Pannek, J. (2017). Occupancy grid based distributed mpc for mobile robots. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 4842–4847. IEEE.

[53] Mendes Filho, J. M. and Lucet, E. (2016). Multi-robot motion planning: a modified receding horizon approach for reaching goal states. *Acta Polytechnica*, 56(1):10–17.

[54] Mendes Filho, J. M., Lucet, E., and Filliat, D. (2017). Real-Time Distributed Receding Horizon Motion Planning and Control for Mobile Multi-Robot Dynamic Systems. In *ICRA 2017 - IEEE International Conference on Robotics and Automation* .

[55] Mendes Filho, J. M., Lucet, E., and Filliat, D. (2018a). Distributed receding horizon motion planning for multirobot systems. https://youtu.be/bpxKhNzhJU8.

[56] Mendes Filho, J. M., Lucet, E., and Filliat, D. (2018b). Experimental Validation of a Multirobot Distributed Receding Horizon Motion Planning Approach. In *ICARCV 2018 - 15th International Conference on Control, Automation, Robotics and Vision*.

[57] Merlhiot, X., Garrec, J. L., Saupin, G., and Andriot, C. (2012). The XDE Mechanical Kernel: Efficient and Robust Simulation of Multibody Dynamics with Intermittent Nonsmooth Contacts. In *Proceedings of the Second Joint International Conference on Multibody System Dynamics - IMSD 2012*.

[58] Milam, M. B. (2003). *Real-time optimal trajectory generation for constrained dynamical systems*. PhD thesis, California Institute of Technology.

[59] Mohanan, M. and Salgoankar, A. (2018). A survey of robotic motion planning in dynamic environments. *Robotics and Autonomous Systems*, 100:171 – 185.

[60] Nedelkoska, L. and Quintini, G. (2018). Automation, skills use and training. (202).

[61] Nilsson, N. J. (1984). Shakey the robot. Technical report, SRI INTERNATIONAL MENLO PARK CA.

[62] Nocedal, J. and Wright, S. J. (1999). *Numerical optimization : with 85 illustrations*. Springer series in operations research. Springer, New York, Berlin, Heidelberg. Tirage corrigé: 2000.

[63] Omron Adept (2016). Omron Adept Lynx Platform User's Guide. https://wiki.metropolia.fi/download/attachments/139693825/Omron%2BAdept%2BLynx%2BPlatform%2BUser_UG_EN_2016_R258IE01.pdf?version=2&modificationDate=1474015160000&api=v2. Accessed: 2019-22-10.

[64] Otte, M. and Frazzoli, E. (2015). Rrtx: Real-time motion planning/replanning for environments with unpredictable obstacles. In *Algorithmic Foundations of Robotics XI*, pages 461–478. Springer.

[65] Park, J.-S. and Oh, S.-J. (2012). A new concave hull algorithm and concaveness measure for n-dimensional datasets. *Journal of Information science and engineering*, 28(3):587–600.

[66] Pedersen, M. R. and Krüger, V. (2015). Automated planning of industrial logistics on a skill-equipped robot. In *IROS 2015 workshop Task Planning for Intelligent Robots in Service and Manufacturing*. Citeseer.

[67] Perez, R. E., Jansen, P. W., and Martins, J. R. R. A. (2012). pyOpt: A Python-based object-oriented framework for nonlinear constrained optimization. *Structures and Multidisciplinary Optimization*, 45(1):101–118.

[68] Piegl, L. and Tiller, W. (2012). *The NURBS book*. Springer Science & Business Media.

[69] Pourboghrat, F. and Karlsson, M. P. (2002). Adaptive control of dynamic mobile robots with nonholonomic constraints. *Computers & Electrical Engineering*, 28(4):241–253.

[70] Quinlan, S. and Khatib, O. (1993). Towards real-time execution of motion tasks. In *Experimental Robotics II*, pages 239–254. Springer.

[71] RATP (2018). Ratp group launches experiment in driverless shuttles at cea paris-saclay. https://www.ratpdev.com/en/newsroom/publications/ratp-group-launches-experiment-driverless-shuttles-cea-paris-saclay. Accessed: 2019-10-03.

[72] RATP (2019). Ratp group, cea and iveco bus hold first ever demonstration in europe of a fully autonomous garage. https://www.uitp.org/sites/default/files/PR%20Autonomous%20Garage%2003302018.pdf. Accessed: 2019-10-03.

[73] Renault, G. (2019). Paris-Saclay Autonomous Lab: new autonomous, electric and shared mobility services. https://shorturl.at/cdkJX. Accessed: 2019-10-03.

[74] Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial intelligence*, 5(2):115–135.

[75] Samson, C. (1995). Control of chained systems application to path following and time-varying point-stabilization of mobile robots. *IEEE Transactions on Automatic Control*, 40(1):64–77.

[76] Sanchez, G. and Latombe, J.-C. (2002). Using a prm planner to compare centralized and decoupled planning for multi-robot systems. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 2, pages 2112–2119. IEEE.

[77] Siciliano, B. and Khatib, O. (2008). *Springer handbook of robotics, pages 808-810*. Springer Science & Business Media.

[78] Siméon, T., Leroy, S., and Lauumond, J.-P. (2002). Path coordination for multiple mobile robots: A resolution-complete algorithm. *IEEE Transactions on Robotics and Automation*, 18(1):42–49.

[79] Spark (2019). Robotics 2020 multi-annual roadmap for robotics in europe. https://www.eu-robotics.net/cms/upload/downloads/ppp-documents/Multi-Annual_Roadmap2020_ICT-24_Rev_B_full.pdf. Accessed: 2019-10-03.

[80] Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., and Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 639–646. IEEE.

[81] Stroyan, K. D. (2014). Symbolic differentiation. In *Calculus Using Mathematica*, pages 85–103. Academic Press.

[82] Takahashi, O. and Schilling, R. J. (1989). Motion planning in a plane using generalized voronoi diagrams. *IEEE Trans. Robotics and Automation*, 5:143–150.

[83] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic robotics (intelligent robotics and autonomous agents series)*. The MIT Press.

[84] Thuilot, B. (1995). *Contribution à la modélisation et à la commande de robots mobiles à roues*. PhD thesis, MINES ParisTech.

[85] Tits, A., Lawrence, C., and Zhou, J. (1999). User's guide for cfsqp version 2.5: A c code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints. *Technical report TR-94-16r1, University of Maryland, College Park*.

[86] Van Parys, R. and Pipeleers, G. (2016). Online distributed motion planning for multi-vehicle systems. In *2016 European Control Conference (ECC)*, pages 1580–1585. IEEE.

[87] Zhou, Y., Hu, H., Liu, Y., Lin, S. W., and Ding, Z. (2017). A Real-Time and Fully Distributed Approach to Motion Planning for Multirobot Systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, PP(99):1–15.

Appendix A

Mathematical programming

A.1 Numerical Optimizers

There is a variety of numerical optimization packages implemented in many different programming languages available for solving optimization problems [67]. Each of them may have their own way of defining the optimization problem and may or may not support specific kinds of constraints (equations, inequations or boundaries).

For the implementation of the DRHMP algorithm made in C++ for using within the physics simulation environment XDE, several libraries were considered.

OPT++ is a library that uses whether OptNIPS, a free nonlinear interior-point algorithm or NPSOL, a licensed sequential quadratic programming algorithm. Both require the user to implement Hessian matrix.

IPOPT (Interior Point OPTimizer) is a software package for large-scale nonlinear optimization. IPOPT implements an interior-point algorithm for continuous, nonlinear, nonconvex, constrained optimization problems. It is meant to be a general purpose nonlinear programming (NLP) solver. However, it is mainly written for large-scale problems with up to million of variables and constraints. IPOPT presents a reasonably easy to use C++ interface but, like the previous library, it requires the implementation of gradients, Jacobians and Hessians for the objective function and constraints. However, a good example code is available on their website that shows how to use the ADOL-C (Automatic Differentiation by OverLoading in C++) package in order to facilitate the evaluation of those first and higher derivatives.

NLopt is a free/open-source library for nonlinear optimization, providing a common interface for a number of different free optimization routines available online as well as original implementations of various other algorithms. Within the NLopt library three methods were applicable to our NLPs:

- ISRES (a global optimizer) that combined with the augmented Lagrangian method could handle nonlinear constraints
- COBYLA which is a local, derivative-free optimizer and as such does not need computation of gradients, Jacobians nor Hessians
- SLSQP a SQP method. SQP methods attempt to solve a nonlinearly constrained optimization problem where the object function and the constraints are twice continuously differentiable. They do so by modeling the object function ($\min f(x)$) at the current iterate x_k by a quadratic programming subproblem and using the minimizer of this subproblem to define a new iterate x_{k+1} [62]. In particular, SLSQP uses the Han–Powell quasi–Newton method with a BFGS update of the B–matrix and an L1–test function in the step–length algorithm. The optimizer uses a slightly modified version of Lawson and Hanson's NNLS nonlinear least-squares solver. It requires derivatives.

RobOptim is a C++ Library for Numerical Optimization applied to Robotics that provides a single interface for various state-of-the-art solvers including IPOPT, NLopt.

Important values for optimization were set as follows:

parameter	meaning	value
`opt_objective_func_abs_tol`	Abs. tolerance on objective function value	10^{-9}
`opt_objective_func_rel_tol`	Rel. tolerance on objective function value	0
`opt_param_rel_tol`	Rel. tolerance on optimization parameters	0
`opt_param_abs_tol`	Abs. tolerance on optimization parameters	0
`opt_equetions_abs_tol`	Abs. tolerance on equations constraints	0
`opt_inequetions_tol_abs_tol`	Abs. tolerance on inequations constraints	0
`num_dif_eps`	Numerical diff step equals $\epsilon 10^{\text{value}}$	5

Appendix B

NCGPC-M

B.1 Law Synthesis

Let us detail the intermediate steps for arriving at the expression of u as showed in Eq. 3.7. As a reminder, we rewrite it here below:

$$\begin{aligned} \frac{\partial J}{\partial u} &= 0_{p\times 1} \\ &\Rightarrow \quad u = -(D^T D)^{-1} D^T (K^{ss})^{-1} (K^s L_y - R^s) \end{aligned}$$

First, we need to develop the expression of the cost function J (by rewriting J_i):

$$\begin{aligned} J_i &= \frac{1}{2}\int_0^{T_i} (y_i - y_{i,\text{ref}})^2 d\tau \\ &= \frac{1}{2}\int_0^{T_i} y_i^2 d\tau - \int_0^{T_i} y_i y_{i,\text{ref}} d\tau + \frac{1}{2}\int_0^{T_i} y_{i\,\text{ref}}^2 d\tau \\ &= \frac{1}{2}\int_0^{T_i} L_{y_i}^T \Lambda_i^T \Lambda_i L_{y_i} d\tau - \int_0^{T_i} y_{i,\text{ref}} \Lambda_i L_{y_i} d\tau \\ &\quad + \underbrace{\frac{1}{2}\int_0^{T_i} y_{i,\text{ref}}^2 d\tau}_{C_i} \\ &= \frac{1}{2} L_{y_i}^T \underbrace{\int_0^{T_i} \Lambda_i^T \Lambda_i d\tau}_{K_i} L_{y_i} - \underbrace{\int_0^{T_i} y_{i,\text{ref}} \Lambda_i d\tau}_{R_i} L_{y_i} + C_i \\ &= \frac{1}{2} L_{y_i}^T K_i L_{y_i} - R_i L_{y_i} + C_i \end{aligned}$$

Now we replace J_i in the expression $\partial J/\partial u$ and develop it aiming to isolate u:

$$
\begin{aligned}
\frac{\partial J}{\partial u} &= \sum_{i=1}^{m}\left(\frac{1}{2}\frac{\partial L_{y_i}^T K_i L_{y_i}}{\partial u} - \frac{\partial R_i L_{y_i}}{\partial u} + \cancelto{0}{\frac{\partial C_i}{\partial u}}\right) \\
&= \sum_{i=1}^{m}\left(\begin{bmatrix} L_{f_{b,1}} L_{f_a}^{(\rho-1)} y_i \\ \vdots \\ L_{f_{b,p}} L_{f_a}^{(\rho-1)} y_i \end{bmatrix} (K_i^s L_{y_i} - R_i^s)\right) \\
&= \underbrace{\begin{bmatrix} L_{f_{b,1}} L_{f_a}^{(\rho-1)} y_1 & \dots & L_{f_{b,1}} L_{f_a}^{(\rho-1)} y_m \\ & \ddots & \\ L_{f_{b,p}} L_{f_a}^{(\rho-1)} y_1 & \dots & L_{f_{b,p}} L_{f_a}^{(\rho-1)} y_m \end{bmatrix}}_{D} \left(\begin{bmatrix} K_1^s L_{y_1} \\ \vdots \\ K_m^s L_{y_m} \end{bmatrix} - \begin{bmatrix} R_1^s \\ \vdots \\ R_m^s \end{bmatrix}\right) \\
&= D\left(\begin{bmatrix} K_1^s L_{y_1} \\ \vdots \\ K_m^s L_{y_m} \end{bmatrix} - \underbrace{\begin{bmatrix} R_1^s \\ \vdots \\ R_m^s \end{bmatrix}}_{R^s}\right) = 0_{p\times 1} \Rightarrow
\end{aligned}
$$

$$
\Rightarrow^{c.\,1} \quad \begin{bmatrix} K_1^s L_{y_1} \\ \vdots \\ K_m^s L_{y_m} \end{bmatrix} - R^s = 0_{m\times 1}
$$

$$
\begin{aligned}
&= K^{ss} D \underbrace{\begin{bmatrix} u_1 \\ \vdots \\ u_p \end{bmatrix}}_{u} + K^s \underbrace{\begin{bmatrix} y_1 \\ \vdots \\ L_{f_a}^{(\rho_1)} y_1 \\ \vdots \\ y_m \\ \vdots \\ L_{f_a}^{(\rho_m)} y_m \end{bmatrix}}_{L_y} - R^s \\
&= K^{ss} D u + K^s L_y - R^s = 0 \Rightarrow \\
&\Rightarrow u = -(D^T D)^{-1} D^T (K^{ss})^{-1} (K^s L_y - R^s)
\end{aligned}
$$

c. 1 refers to the first condition for nonlinear control-affine MIMO system from section 3.3.3.

B.2 Running the Controller in Real-time

For achieving desired performance it is important to exploit the real-time features of processes under Linux. We installed a real-time kernel and run our controller with correct priority and affinity settings.

Appendix C

HPN

$$
\mathbf{bs_0} = \left\{
\begin{array}{l}
\text{obj}_A = \left\{
\begin{array}{l}
\mu = [-0.14,\ -0.14,\ 0.0], \\
\sigma = [0.0003,\ 0.0003,\ 0.0003], \\
\text{shape} = \text{"rectangle"}, \\
\text{dimension} = [0.14,\ 0.17], \\
\text{color} = \text{"\#41f4b8"}, \\
\text{movable} = \text{True}
\end{array}
\right. \\
\text{obj}_B = \left\{
\begin{array}{l}
\mu = [0.14,\ -0.14,\ 0.0], \\
\sigma = [0.0003,\ 0.0003,\ 0.0003], \\
\text{shape} = \text{"rectangle"}, \\
\text{dimension} = [0.14,\ 0.17], \\
\text{color} = \text{"\#41f4b8"}, \\
\text{movable} = \text{True}
\end{array}
\right. \\
\text{obj}_C = \left\{
\begin{array}{l}
\mu = [0.14,\ 0.14,\ 0.0], \\
\sigma = [0.0003,\ 0.0003,\ 0.0003], \\
\text{shape} = \text{"rectangle"}, \\
\text{dimension} = [0.14,\ 0.17], \\
\text{color} = \text{"\#f48042"}, \\
\text{movable} = \text{True}
\end{array}
\right. \\
\text{obj}_D = \left\{
\begin{array}{l}
\mu = [-0.14,\ 0.14,\ 0.0], \\
\sigma = [0.0003,\ 0.0003,\ 0.0003], \\
\text{shape} = \text{"rectangle"}, \\
\text{dimension} = [0.14,\ 0.17], \\
\text{color} = \text{"\#e8d75c"}, \\
\text{movable} = \text{True}
\end{array}
\right. \\
\text{immov} = \left\{
\begin{array}{l}
\mu = [0.0,\ 0.35,\ 0.0], \\
\sigma = [0.0003,\ 0.0003,\ 0.0003], \\
\text{shape} = \text{"rectangle"}, \\
\text{dimension} = [0.6,\ 0.1], \\
\text{color} = \text{"\#282828"}, \\
\text{movable} = \text{False}
\end{array}
\right.
\end{array}
\right.
$$

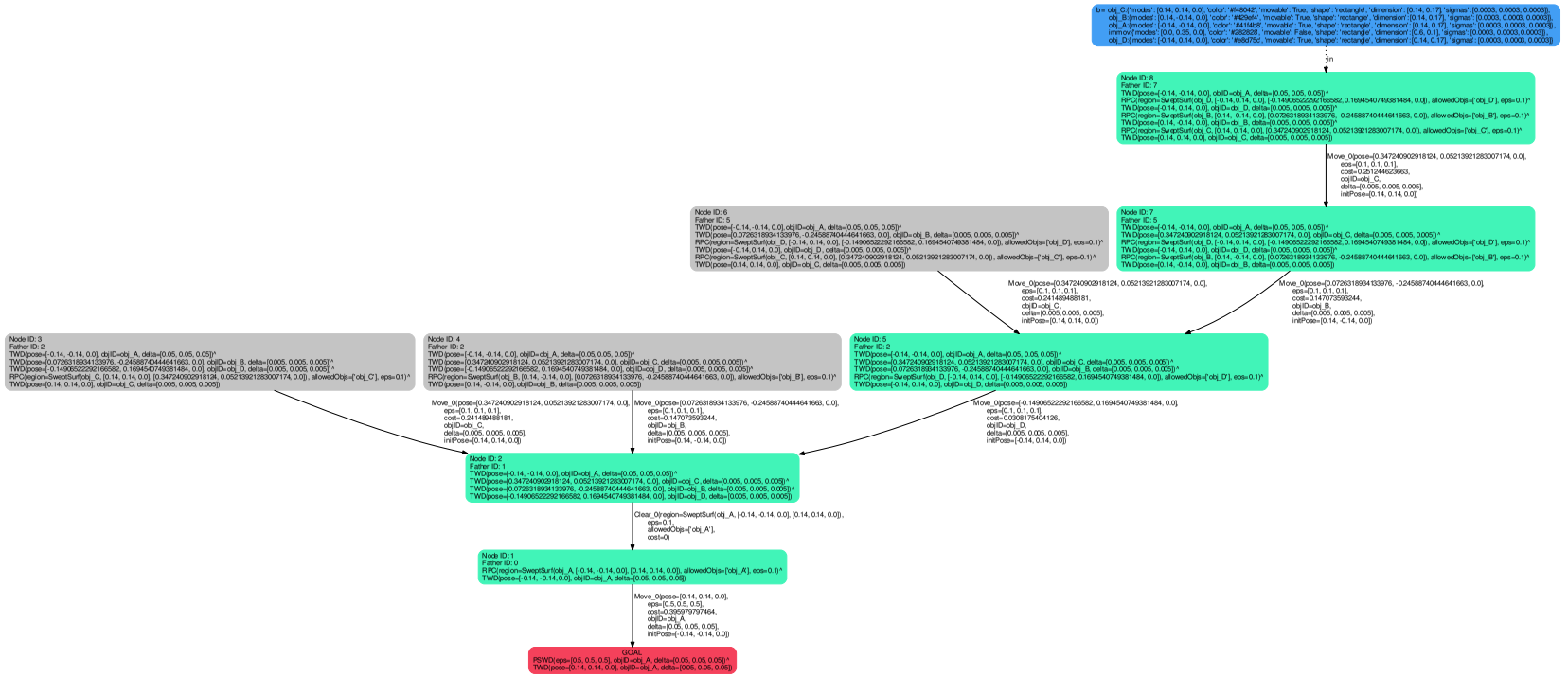


Fig. C.1 Detailed version of Fig. 4.9

Fig. C.2 Detailed version of Fig. 4.12

INSTITUT POLYTECHNIQUE DE PARIS

Titre : Planification de Mouvements en Ligne et Distribuée de Systèmes Multi-Robots Mobiles

Mots clés : Robots Mobiles, Systèmes Multi-robot, Planification de Trajectoire, Optimisation

Résumé :
L'objectif de cette thèse est d'étudier et de développer une approche pour résoudre le problème de planification de mouvements d'un groupe de robots mobiles à roues en environnement opérationnel réaliste. Nous proposons principalement une approche basée sur l'optimisation distribuée associée à une méthode de type fenêtre glissante pour la génération de trajectoire en boucle ouverte ainsi qu'une loi de commande prédictive (MPC) pour la stabilisation du système en boucle fermée. Dans cette approche, la perception, la planification de trajectoire et son exécution sont combinées et peuvent être réalisées par le contrôleur de chacun des robots indépendamment, au fur et à mesure qu'ils évoluent dans leur espace de travail. L'approche garantit le respect de plusieurs types de contraintes, à savoir l'évitement des obstacles, la limitation des vitesses et des accélérations, les contraintes non-holonomes et l'évitement des collisions inter-robots. Les robots appartenant au système multi-robots échangent des informations sur leurs trajectoires envisagées et convergent individuellement vers des trajectoires optimales sans conflit.

En outre, des travaux en vue d'une planification intégrée des tâches et des mouvements par une méthode hiérarchique sont présentés. L'objectif étant d'aboutir à une méthode complète de planification de mouvements robuste et hautement autonome des robots mobiles.
Des expériences en simulation et avec des véhicules réels de type monocycle non-holonomes ont été menées. Elles ont permis d'analyser l'impact des paramètres sur des critères déterminants tels que le temps de calcul, l'évitement des obstacles, l'évitement des collisions inter-robots et le temps de déplacement. Ces résultats démontrent également la qualité du mouvement du robot dans des situations où la dynamique, les incertitudes sur la localisation du robot et les délais de communication sont réels et significatifs.
Finalement, cette étude montre que l'approche proposée pourrait être utilisée dans des systèmes réels où l'incertitude sur l'état de l'environment, les retards de communication, la puissance de calcul embarquée limitée, la forte dynamique et d'autres phénomènes habituellement difficiles à surmonter sont tous présents.

Title: Online Distributed Motion Planning for Mobile Multi-robot Systems

Keywords: Mobile Robots, Multi-robot Systems, Trajectory Planning, Mathematical Programming

Abstract:
This thesis aims to study and develop an approach for solving the motion planning problem of a group of wheeled mobile robots in a realistic environment. Mainly we propose a distributed mathematical programming approach associated with a receding horizon method for the open-loop trajectory generation as well as a modified model predictive control (MPC) for the closed-loop stabilization. In this approach, perception, trajectory planning, and execution are interleaved and can be performed onboard each robot independently, as they evolve through their workspace. It ensures respect of several types of constraints, namely obstacle avoidance, bounded velocities and accelerations, nonholonomic constraints, and inter-robot collision avoidance. The robots belonging to the multi-robot system exchange information on their intended trajectories and converge individually to optimal non-conflicting trajectories.

Furthermore, some work towards integrated task and motion planning by a hierarchical method is presented. The objective was to achieve a complete framework for robust, highly autonomous mobile robot motion.
Experiments both in simulation and with real nonholonomic unicycle-like vehicles were conducted. They allowed us to analyze the impact of parameters on key figures such as computation time, obstacle avoidance, inter-robot collision avoidance, and travel time. Results also show the quality of robot motion in situations where dynamics, the uncertainties about robot localization, and communication delays are real and meaningful.
Overall, this study indicates that the proposed approach could be used in real systems where uncertainty about the world state, communication delays, limited onboard computation power, strong dynamics, and other usually challenging to overcome phenomena are all present.

Institut Polytechnique de Paris
infos@ip-paris.fr
ENSTA Paris
828 Boulevard des Maréchaux, 91120 Palaiseau