Czech Technical University in Prague
Faculty of Electrical Engineering

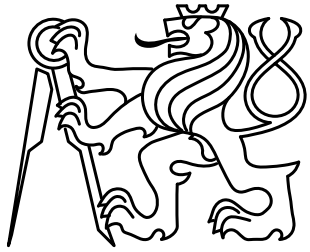
Doctoral Thesis

December 2016

Michal Čáp

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science



CENTRALIZED AND DECENTRALIZED ALGORITHMS FOR MULTI-ROBOT TRAJECTORY COORDINATION

Doctoral Thesis

Michal Čáp

Prague, December 2016

Ph.D. Programme: Electrical Engineering and Information Technology
Branch of study: Artificial Intelligence and Biocybernetics

Supervisor: Prof. Dr. Michal Pěchouček, MSc.

Co-supervisor: Dr. rer. nat. Peter Novák

Dedicated to my brothers David, Vojta, and Sláva.

Acknowledgments

This thesis would not be possible without the help of many people. Foremost, I would like to thank the three senior researchers that provided me with support, encouragement, and guidance: my supervisor Michal Pěchouček for allowing to undergo my PhD endeavor in his research center, which proved to be a fertile and highly inspiring environment; my co-supervisor Peter Novák for being always present to provide critical feedback to my work; and Jiří Vokřínek whose encouragement and support allowed me to fully focus on my research.

I am also grateful to all my coworkers, in particular to Antonín Komenda, Viliam Lisý, Ondřej Vaněk, Michal Štolba, Malcolm Egan, and Slava Kungurtsev for many fruitful discussions and the feedback that helped me to shape my work. I am greatly thankful to the Fulbright Commission and CTU in Prague for allowing me to spend one academic year at MIT and to prof. Emilio Frazzoli and his group for accepting me in their research group and for collaborating with me. Special thanks also go to my collaborator Alexander Kleiner, who introduced me to the core robotics community and frequently reminded me to mind the gap between the theoretical models and the requirements of real robotic systems.

Most importantly, I would like to thank my parents and the rest of my family for raising me and supporting me up to this point in my life and especially my wife Martina for her endless support and tolerance along the long journey leading to this dissertation.

Abstract

One of the standing challenges in multi-robot systems is how to reliably avoid collisions among individual robots without jeopardizing the mission of the system. This is because the existing collision-avoidance techniques are either *prone to deadlocks*, i.e., the robots may never reach their desired goal position, or *computationally intractable*, i.e., the solution may not be provided in practical time. We study whether it is possible to design a method for collision avoidance in multi-robot systems that is both deadlock-free and computationally tractable. The central results of our work are 1) the observation that in appropriately structured environments deadlock-free *and* computationally tractable collision avoidance is, in fact, possible to achieve and 2) consequently we propose practical, yet guaranteed, centralized and decentralized algorithms for collision avoidance in multi-robot systems.

We take the deliberative approach, i.e., coordinated collision-free trajectories are first computed either by a central motion planner or by decentralized negotiation among the robots and then each robot controls its advancement along its planned trajectory. We start by reviewing the existing techniques in both single- and multi-robot motion planning, identify their limitations, and subsequently design new centralized and decentralized trajectory coordination algorithms for different use cases.

Firstly, we prove that a revised version of the classical prioritized planning technique, which may not return a solution in general, is guaranteed to always return a solution in polynomial time under certain conditions that we characterize. Specifically, it is guaranteed to provide a solution if the start and destination of each coordinated robot is an endpoint of a so-called *well-formed infrastructure*. That is, it can be reliably used in systems where the robots at start and destination positions do not prevent other robots from reaching their goals, which, notably, is a property satisfied in most man-made environments.

Secondly, we design an asynchronous decentralized variant of both classical and revised prioritized planning that can be used to find coordinated trajectories solely by peer-to-peer message passing among the robots. The method inherits guarantees from its centralized version, but can compute the solution faster by exploiting the computational power distributed across multi-robot team.

Thirdly, in contrast to the above algorithms that coordinate robots in a batch, we design a decentralized algorithm that can coordinate the robots in the systems incrementally. That is, the robots may be ordered to relocate at any time during the operation of the system. We prove that if the robots are tasked to relocate between endpoints of a well-formed infrastructure, then the algorithm is guaranteed to always find a collision-free trajectory for each relocation task in quadratic time.

Fourthly, we show that incremental replanning of trajectories of individual robots while they are subject to gradually increasing collision penalty can serve as a powerful heuristic that is able to generate near-optimal solutions.

Finally, we design a novel control law for controlling the advancement of individual robots in the team along their planned trajectories in the presence of delaying disturbances, e.g., humans stepping in the way of robots. While naive control strategies for handling the disturbances may lead to deadlocks, we prove that under the proposed control law, the robots are guaranteed to always reach their destination.

We evaluate the presented techniques both in synthetic simulated environments as well as in real-world field experiments. In simulation experiments with up to 60 robots, we observe that the proposed technique generates shorter motions than state-of-the-art reactive collision avoidance techniques and reliably solves also the instances where reactive techniques fail. Further, unlike many proposed coordination techniques, we validate the assumptions of our algorithms and the consequent practical applicability of our approach by implementing and testing proposed coordination approach in two real-world multi-robot systems. In particular, we successfully deployed and field tested asynchronous decentralized prioritized planning as a collision avoidance mechanism in 1) a Multi-UAV system with fixed-wing unmanned aircraft and 2) an experimental mobility-on-demand system using self-driving golf carts.

Contents

Chapter 1

Introduction

Building on the recent advances in robotics and automation, both industry and academia increasingly focus on the development of different kinds of autonomous multi-vehicle systems ranging from household cleaning robots and industrial intra-logistic systems to large-scale globally-operating transport systems. In most of these systems, one of the important concerns is how to ensure safe and reliable coordination of motions of individual autonomous robots. Some example application areas with anticipated use of multiple mobile robots that have to be coordinated in order to avoid collisions are the following:

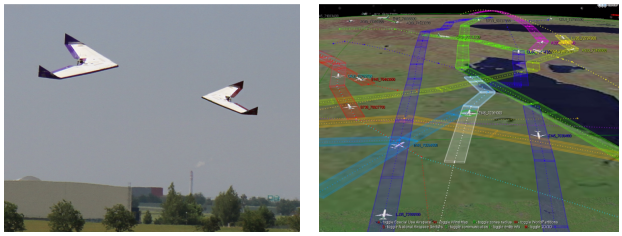
(1) Household cleaning robots such as the iRobot's Roomba are a common sight in today's homes. With their prices further dropping, it can be expected that in near future it will not be unusual to see several such devices operating in a single household. Therefore, the new models will have to be able to gracefully cope with encounters with other robots performing their duties on the same floor.

(2) Initiatives such as Industry 4.0 [GTAI, 2014] call for a paradigm shift from today's rigid centralized manufacturing to flexible decentralized manufacturing process allowing rapid reconfiguration of production capacities. The fourth generation manufacturing consists of interconnected autonomous production assets that dynamically team-up to produce each individual product. One of the crucial building blocks in such a future factory are flexible intralogistics systems based, e.g., on fleets of mobile robots designed to autonomously transport manufactured products between workstations. A recent example of such a system is an unconventional warehouse management system developed by Kiva systems (now Amazon Robotics) that employs a large cooperative team of autonomous mobile robots to move around storage shelves as needed [Wurman et al., 2007]. Such systems typically consist of a large number of mobile robots operating in a confined area which requires close coordination of motions of individual robots.

(3) Another field that has seen a lot of progress recently is the autonomous driving. Traditional car manufacturers, IT companies, as well as academic institutions invest heavily in developing prototypes of self-driving cars [Davies, 2015, Rosen, 2012, Lardinois and Wilhelm, 2015]. Some of the players progress incrementally and develop more and more sophisticated driving assistance, while others attempt to develop fully autonomous cars from scratch. There are car manufacturers that predict that fully autonomous cars will be available on the market by 2020 [Mack, 2014]. If such promises are met, a significant fraction of the cars on the roads will be in a near future autonomous and consequently each autonomous vehicle will have to come with a capability to autonomously coordinate its intended trajectory with the trajectories of other autonomous cars in order to prevent collisions and ensure progress towards their goals even in challenging, unstructured traffic scenarios.

(4) Yet another example of a successful deployment of results from robotics research is the growing UAV industry. Different models of semi-autonomous UAVs are now available as off-the-shelf products and both academia and industry are investigating possibility to employ autonomous UAVs for tasks such as autonomous logistics [D'Onfro, 2014] or real-time area surveillance with multiple flying assets [Selecký et al., 2013]. An important consideration in such systems is again how to coordinate the

(a) Household robots (b) Industrial intralogistics (c) Autonomous cars



(d) Multi UAV system (e) Next generation air-traffic management

Figure 1.0.1: Example application areas requiring coordination of multiple autonomous robots. The picture of household robots courtesy of Justin Dolske.

trajectories of all involved vehicles so as to avoid collisions between the individual flying assets.

(5) Finally, there is a growing concern that the current method of air-traffic management will not sustain the predicted future growth of air-traffic intensity [FAA, 2015]. Therefore, the regulators such as FAA or EUROCONTROL are exploring new paradigms for air-traffic control. One of the considered options is a so-called free-flight concept [Leslie, 1996] in which the individual airplanes are not subject to a supervision of a ground air-traffic control center, but instead, they fly freely in a similar manner to cars driving on the roads today. The conflicts between planned paths of individual airplanes are detected by a dedicated on-bard device and resolved by an automated negotiation between the airplanes. Clearly, a major challenge in such an application is the design of a guaranteed negotiation protocol, i.e., a decentralized algorithm that will be able to provably resolve all possible conflicts.

All the above-described application areas, illustrated also in Figure 1.0.1, represent examples of multi-vehicle autonomous systems, in which one of the crucial requirements is the capability of individual vehicles to *coordinate their trajectories in order to prevent their mutual collisions*. In this thesis, we will survey existing approaches and develop novel algorithmic methods that can be used to coordinate motions in such multi-robot systems.

1.1 Problem Statement and Existing Methods

A practical coordination mechanism for large-scale decentralized multi-robot systems, as exemplified by the application domains in the previous section, should satisfy several key requirements: it should guarantee the ability to provide a solution, be computationally tractable, provide solutions of acceptable quality, be generally applicable to different types of robots, and suitable for decentralized implementation. We now discuss the individual requirements in detail.

Guaranteed solution: It is highly desirable to know whether and under what conditions is the particular coordination mechanism guaranteed to provide a solution that will lead all the robots to their destinations. An application of a coordination mechanism that does not guarantee an ability to provide

such a solution may result in deadlocks and other failure modes requiring an intervention of a human operator, which negatively affects the ability of the system to operate autonomously.

Tractability: Since most of the considered multi-robot systems are expected to respond to newly assigned tasks interactively in the order of seconds, the coordination mechanism must be able to compute and provide a solution reasonably fast even for large multi-robot teams. As we will see later, the guaranteed trajectory coordination methods suffer from worst-case time complexity that is exponential in the number of coordinated robots, which makes them inapplicable to systems with more than a few robots. It is therefore desired that the time complexity of the algorithm scales polynomially in the number of robots.

Generality: Different shapes and locomotion mechanisms of individual robots impose different constraints on the relative positions of robots and the trajectories that the robots can follow. A practically applicable coordination method must be flexible enough to allow modeling of such real-world constraints.

Quality: An ideal coordination mechanism would find an optimal solution, e.g., such that leads robots to their destination in minimum total time. However, such solutions can be prohibitively hard to compute. In many systems, it is acceptable to relax the requirement on the optimality, but the quality of the resulting coordinated motion should still be reasonable. For example, the robots should proceed towards their destinations in parallel when possible, the robots should not stop for longer than needed, the gap between the robots passing each other should not be unnecessarily large, etc.

Decentralization: In heterogeneous multi-robot teams, it is often more desirable to coordinate individual robots in a decentralized manner, either by letting the robots observe each other's actions or by letting them communicate with each other. In a centralized system, all robots must formalize and communicate their task specification and mobility constraints to a central solver, before the central solver can compute the coordinated trajectories for the robots. In contrast, the decentralized approach allows a task specification and constraints to remain private to each robot. This enables the design of very simple, but general coordination protocols that can be readily used across various types of autonomous assets. Further, a decentralized solution is useful in situations when the robots in a multi-robot system are owned and controlled by different entities, where each entity is interested in minimizing the disclosure of private information.

Existing Approaches

In this section, we provide a brief overview of existing multi-robot coordination techniques and their properties. The problem of coordinating motions of multiple robots can be approached either from a control engineering perspective by employing the *reactive paradigm* to collision avoidance or from an AI perspective by employing the *deliberative paradigm.*

In the *reactive paradigm*, each robot follows the shortest path to its current destination and attempts to resolve collision situations locally as they appear. Each robot periodically observes positions and velocities of other robots in its neighborhood and if it detects a potential future collision, the robot attempts to avert the collision by adjusting its immediate heading and velocity. A number of methods have been proposed [van den Berg et al., 2011, Guy et al., 2009, Alonso-Mora et al., 2013] that prescribe how should be such a collision-avoiding velocity computed in a reciprocal multi-robot setting. These approaches are widely used in practice thanks to their computational efficiency (a collision-avoiding velocity for a robot can be computed in a fraction of a millisecond [van den Berg et al., 2011]) and inherently decentralized nature. However, since the reactive approaches resolve collisions only locally, they are prone to deadlocks and potentially inefficient. That is, the robots are not guaranteed to reach their destination and the resulting motions can be much longer than necessary. The reactive techniques often provide satisfactory performance in uncluttered environments, but their applicability in confined environments is limited.

In the *deliberative paradigm*, the system first finds a solution to a multi-robot trajectory coordination problem, i.e., it plans a set of globally coordinated collision-free trajectories from the origin to the

destination of each robot. After the coordinated trajectories have been found, the robots start following their respective coordinated trajectories. If the robots execute the resulting joint plan precisely (or within some predefined tolerance), it is guaranteed that they will reach their destination while avoiding collisions with other robots. The central challenge in the deliberative approach is therefore how to efficiently obtain a solution to the trajectory coordination problem. A multi-robot trajectory coordination problem is typically formulated as follows: Consider n robots indexed $1, ..., n$ operating in 2-d or 3-d static fully-observable environment with obstacles. The trajectory of each robot must start at its initial position, reach its goal position, and respect the given kinematic/dynamic constraints on the motion of the robot. The task is to find a set of trajectories, one for each robot, such that a) the total time the robots spend in transit between the initial and goal position (or some other cost metric) is minimal and b) the robots do not collide when executing the found trajectories. The early theoretical results show that even the simplest, satisfying (i.e., non-optimal) variant of the problem is intractable. More precisely, coordination of rectangles is PSPACE-hard [Hopcroft et al., 1984] and coordination of disks is NP-complete [Schwartz and Sharir, 1983b].

A straightforward *coupled* solution approach to the problem is to view the individual robots as one coupled robot and use some standard motion planning algorithm to find a solution. However, the computational complexity of such an approach grows exponentially with the number of considered robots and thus this approach quickly becomes impractical when one attempts to plan for more than a few robots. Further, this approach is difficult to implement in a decentralized way.

In an alternative abstracted formulation of the problem, referred to as *pebble motion problem*, the robots are imagined as "pebbles" moving on a graph. Each robot is assumed to occupy exactly one vertex on the given graph and the task is two find a sequence of moves for each robot from its start vertex to its goal vertex such that two robots never occupy one vertex at the same time step. The formalization is however not well suited for modeling of fine-grained motions or systems where robots have different sizes. Although non-optimal solutions to pebble motion problems can be found in polynomial time [de Wilde et al., 2014, Surynek, 2009a], the resulting solutions are sequential and thus significantly sub-optimal in terms of travel time. In fact, the problem of finding distance- and time- optimal coordinated paths in this formulation remains NP-hard [Yu, 2016, Yu and LaValle, 2013].

For systems with many robots that cannot be appropriately modeled in the pebble motion framework, the solution to trajectory coordination problem can be computed using *decoupled* heuristic approaches. The most common decoupled approach is based on the idea of prioritized planning [Erdmann and Lozano-Pérez, 1987]. In prioritized planning, the trajectories for the robots are planned sequentially in the order defined by the robots' "priorities". In each iteration, the system plans a trajectory for a single robot that avoids collisions with higher-priority robots which follow trajectories generated in previous iterations. Due to its decoupled nature, this technique can be decentralized in a relatively straightforward fashion [Čáp et al., 2015b]. Although practical in terms of computational complexity, this approach suffers from incompleteness, i.e., in some situations it fails to find a solution even if one exists.

The properties of existing methods for trajectory coordination in multi-robot teams are summarized in Table 1.1. As we can see, none of the existing techniques combines all the properties needed for practical deployment in large-scale multi-robot systems.

1.2 Research Objectives

Although the research fields of collision-avoidance and multi-robot trajectory coordination have received significant attention in the last three decades and a number of algorithms have been proposed for the problem, for a wide range of application scenarios the existing methods still do not offer satisfactory performance: they rely on assumptions that are hard to satisfy, cannot be relied upon, or suffer from computational complexity that prohibits practical deployment. This problem is particularly pressing in scenarios with many mobile robots operating in an unstructured environment (i.e., the robots do not

	Guaranteed solution?	Tractability	Generality	Solution quality	Decentralization
Reactive techniques	No	Tractable	General	Acceptable in open space, Poor in clutter	Suited
Coupled methods	Yes	Intractable	General	Optimal	Not suited
Pebble motion	Yes	Tractable	Not general	Poor, Sequential solution	Not suited
Decoupled methods	No	Tractable	General	Suboptimal acceptable	Suited
RPP/AD-RPP (Proposed)	Yes (in well-formed infrastructures)	Tractable (in well-formed infrastructures)	General	Suboptimal acceptable	Suited

Table 1.1: Overview of solution techniques for trajectory coordination

follow fixed pre-designed paths) and when the motion of robots is constrained by complicated obstacles that may form challenging environmental features such as narrow corridors. The situation is further complicated by the strong preference for a decentralized solution in many multi-robot applications.

This deficiency became apparent during the initial stages of my PhD study when I encountered the need to address the problem of practical, yet reliable collision avoidance in several applied research projects: Tactical AgentFly [Selecký et al., 2013], for example, was a research project concerned with development of a system allowing high-level tasking and control of multiple aerial vehicles to autonomously carry out tasks such as area surveillance or target tracking. One of the concerns that arose within the project was how to ensure that the flight paths of the aerial vehicles will be conflict-free. Since the permanent communication link between the aerial vehicles and the ground station was often unavailable and the airplanes made most of the decision-making on-board, the conflicts between the future flight plans of different airplanes had to be reliably resolved by peer-to-peer negotiation between the airplanes, while respecting the forbidden "no-fly" zones.

When I later visited Singapore-MIT Alliance for Research and Technology research institute, where the local team of researchers was developing an automated system for passenger transportation [Chong et al., 2012], I discovered that they also have to regularly deal with non-trivial conflicts in planned trajectories of the individual vehicles, since their operational environment contained narrow passages and bridges that could fit only a single vehicle. Their so-called automated mobility-on-demand system consisted of a number of self-driving golf carts designed to be able to autonomously and independently navigate within the NUS university campus. In result, the conflicts between planned trajectories of different vehicles had to be resolved in a decentralized fashion by direct local communication between the vehicles.

Another source of motivation for my work came from Alexander Kleiner who had experience with development of industrial intralogistics systems. I learned that the same challenge exists also within next-generation intralogistics systems. These systems employ mobile robots moving in unstructured environments shared with human workers. The robots' motions need to be reliably coordinated, although, in an industrial setting, they can often be coordinated centrally.

At the time, the approaches that could be considered for collision-avoidance in multi-robot systems were the reactive techniques, coupled multi-robot motion planning, pebble-motion methods, and decoupled multi-robot planning (see Table 1.1). The reactive techniques worked nicely in uncluttered environments but failed to solve many commonly occurring situations, e.g., when two robots met heads-on in a narrow corridor. The coupled approach did not scale beyond systems with only a few robots and furthermore it was difficult to implement in a decentralized manner. The highly abstracted model used in pebble motion methods was not expressive enough to model real-world multi-robot systems. Finally, the decoupled approach was applicable and suited for decentralization, but the method would often fail to solve given conflict scenario – consequently the operation of the system must be interrupted and resolved by a human operator, which undermines the initial motivation for the introduction of automation.

The common concern within all above robotic applications was therefore whether it is possible to design a practically applicable centralized (or, more often, decentralized) method that would *guarantee* the ability to resolve all conflicts between planned trajectories of robots in the system. This became the topic of my dissertation research. More specifically, motivated by the above real-world application needs, the general objective of this dissertation was to study the multi-robot coordination problem and design trajectory coordination algorithms for large multi-robot teams that satisfy the requirements from Section 1.1, i.e., algorithms that are guaranteed, tractable, general, provide solutions of acceptable quality, and suit decentralized systems.

1.3 Achievements

This thesis summarizes the results of research effort driven by the above objective. The following is the list of the major achievements of this thesis:

1. A survey of the state of the art in the single-robot and multi-robot motion planning.
2. Formalization of the concept of a well-formed infrastructure that is used to characterize a practically-relevant tractable fragment of multi-robot trajectory coordination problem and consequently enables the design of tractable guaranteed coordination algorithms for this fragment.
3. Tractable guaranteed algorithms for trajectory coordination in well-formed infrastructures:
   (a) A centralized algorithm for trajectory coordination problem called *Revised Prioritized Planning* scheme (**RPP**) and formulation of a sufficient condition under which the algorithm is guaranteed to provide a solution. The algorithm comes with guarantees, but remains as scalable as classical prioritized planning, i.e., it can compute coordinated motions for tens of robots in the order of seconds.
   (b) An *Asynchronous Decentralized variant* of both classical and revised Prioritized Planning scheme (**AD-(R)PP**) that is guaranteed to terminate and inherits completeness properties from the respective centralized counterpart. We experimentally show that asynchronous decentralized algorithm exhibits better utilization of the distributed computational resources and thus provides up to 2x-faster convergence compared to the previously known synchronized decentralized approach.
   (c) An on-line version of the prioritized approach called *Continuous Best Response Approach* (**COBRA**) that can be used to efficiently coordinate motions in multi-robot systems, where relocation task appear incrementally during the operation of the system. For multi-robot systems operating in well-formed infrastructures, we show that the method is guaranteed to find coordinated collision-free trajectory for every relocation task in polynomial time. Further, our experiments demonstrate that in large multi-robot teams, the method is both more reliable and more efficient than a state-of-the-art reactive collision avoidance method.

(d) A decoupled approach called *k-step Penalty Method* (**kPM**) that can be used as a heuristic able to generate higher quality paths than the ones found by prioritized planning. Experimentally, we found that the coordinated trajectories generated by the kPM algorithm are near-optimal. At the same time, the experiments show that the method offers qualitatively better scalability than an existing state-of-the-art optimal method.

4. A *Robust Multi-Robot Trajectory Tracking Strategy* (**RMTRACK**) that ensures that a multi-robot system executing preplanned coordinated trajectories remains provably deadlock-free and collision-free even in the presence of exogenous disturbances. In our experiments, we found that the proposed method scales significantly better than the baseline deadlock-free approach and furthermore it handles high-intensity delaying disturbances more reliably and more efficiently than an existing state-of-the-art reactive collision avoidance method.

The results reported in this thesis were published in major robotics and industrial automation journals and in top-tier robotics and artificial intelligence conferences. Namely, the survey on single-robot motion planning was written during my year-long visit at MIT and invited for publication for IEEE Transactions on Intelligent Vehicles [Paden, Čáp, Yong, Yershov, and Frazzoli, 2016], a newly established IEEE journal focusing on intelligent and self-driving vehicles. The notion of well-formed infrastructure, RPP algorithm, and AD-(R)PP algorithm were published in IEEE Transactions on Automation Science and Engineering [Čáp et al., 2015b]. The algorithm COBRA was presented at ICAPS conference [Čáp et al., 2015c] and RMTRACK was accepted to IEEE IROS conference [Čáp et al., 2016]. Finally, kPM algorithm was presented at multi-robot coordination workshop at RSS conference [Čáp et al., 2015a]. All of the presented algorithms were extensively evaluated in simulation and some of the algorithms were field-deployed as motion coordination technique in two real-world outdoor multi-robot systems: a system of autonomous aerial vehicles and a system of self-driving cars. The preprints of all the above publications are available at ArXiv preprint server. The implementations of all proposed algorithms and the benchmark instances are freely available at Github.

1.4 Organization

This thesis is organized as follows. The state of the art in both single-robot and multi-robot motion planning is reviewed in Chaper 2. Chapter 3 then states several different flavors of trajectory coordination problem considered in this work. In Chapter 4, we introduce revised prioritized planning (RPP) and prove its completeness in well-formed infrastructures. The asynchronous decentralized variant of (revised) prioritized planning (AD-RPP) is discussed in Chapter 5. The algorithm for incremental coordination called COBRA is introduced in Chapter 6 and the k-step Penalty Method (kPM) is discussed in Chapter 7. The issue of robust execution of multi-robot plans in the presence of disturbance is treated in Chapter 8, where we also introduce the RMTRACK control strategy. In Chapter 9, we report on the deployment of the proposed techniques in two real-world multi-robot systems. Finally, Chapter 10 concludes the thesis.

Chapter 2

Survey

Multi-robot motion planning is a generalization of a more widely studied problem of single-robot motion planning. Moreover, many of the algorithms for multi-robot motion coordination solve a single-robot motion planning problems as a subroutine. Therefore, we will start our exposition in Section 2.1 by reviewing the state of the art in single-robot motion planning. We formulate the problem of path planning and trajectory planning for a single robot, discuss relevant computational complexity results, and overview existing solution methods for the two related single-robot motion planning problems. Anyone familiar with the field can skip this section and continue directly to Section 2.2, where we discuss the problem of multi-robot coordination for collision avoidance, overview the main complexity results and describe the popular solution approaches to the problem.

2.1 Single-robot Motion Planning

The problem of single-robot motion planning can be formulated as follows. Consider an articulated robot that operates in a 2-d or 3-d environment. Let $d \in \{2, 3\}$ be the dimension of the environment. The obstacle-free part of the environment $\mathbb{R}^d$ is called workspace, denoted by $\mathcal{W}$ and defined as $\mathcal{W} = \mathbb{R}^d \setminus \mathcal{O}$, where $\mathcal{O} \subset \mathbb{R}^d$ is the obstacle region, i.e., the set of all points in obstacles. The pose of the robot can be completely described by its *configuration*, which is an n-tuple that specifies its state along all the degrees of freedom of the robot. The set of all feasible configurations of a robot is called the robot's configuration space and denoted by $\mathcal{X}$. For example, the pose of a rectangular robot operating in a 2-d workspace can be described by taking the Cartesian coordinates of a distinguished anchor point of the robot and the orientation of the robot. For such a robot, the robot's configuration is a triplet (p_x, p_y, ϕ), where $(p_x, p_y) \in \mathbb{R}^2$ is the position of the anchor point and $\phi \in [-\pi, \pi]$ is the orientation of the robot with respect to the x-axis. Clearly, the configuration space of such a robot is then $\mathbb{R}^2 \times [-\pi, \pi]$. One can also consider complicated articulated robots where the configuration would consist of the position and orientation of its base and the angle of each joint of the robot.

The configuration of a robot then completely determines the region of the environment that the robot occupies. Let $R : \mathcal{X} \to \mathcal{P}(\mathbb{R}^d)$, where $\mathcal{X}$ is the configuration space of the robot and $\mathcal{P}(S)$ denotes the power set of S, be a function that maps a configuration of the robot to the region of the workspace that the robot occupies at the given configuration. The set of collision-free configurations can be then defined as $\mathcal{X}_{\text{free}} := (\mathbf{x} : \mathbf{x} \in \mathcal{X} \text{ and } R(\mathbf{x}) \subset \mathcal{W})$. Finding a collision-free motion for an articulated robot then amounts to finding a path in the collision-free configuration set $\mathcal{X}_{\text{free}}$ from the given initial configuration $\mathbf{x}_{\text{init}}$ to some goal region X_{goal}. Often, the motions of a particular robotic platform are limited by kinematic and dynamic constraints, which in turn translate to constraints on the feasible path in the configuration space. For instance, a car-like robot can only follow a path with bounded curvature.

A motion plan for a single robot can take the form of a path or a trajectory. Within the path planning

framework, the solution is a *path* represented as a function $\sigma : [0,1] \to \mathcal{X}$, where $\mathcal{X}$ is the configuration space of the robot. Note that such a solution does not prescribe how this path should be followed in time and one can either choose a velocity profile for the path or delegate this task to a low-level controller. Within the trajectory planning framework, the time is explicitly considered, which allows for direct modeling of robot's dynamics and dynamic obstacles. In this case, the solution is a *trajectory* represented as a time-parameterized function $\pi : [0,T] \to \mathcal{X}$, where T is the planning horizon and $\mathcal{X}$ is the configuration space. Unlike a path, the trajectory prescribes how the configuration of the robot evolves over time.

In the following two sections, we provide a formal problem definition of the path planning and trajectory planning problems and review the main complexity and algorithmic results for both formulations.

2.1.1 Path Planning

The path planning problem is to find a path $\sigma : [0,1] \to \mathcal{X}$ in the configuration space $\mathcal{X}$ of the robot that starts at the initial configuration and reaches the goal region while satisfying given global and local constraints. Depending on whether the quality of the solution path is considered, the terms *feasible* and *optimal* are used to describe this path. Feasible path planning refers to the problem of determining a path that satisfies some given problem constraints without focusing on the quality of the solution; whereas optimal path planning refers to the problem of finding a path that optimizes some quality criterion subject to given constraints.

The optimal path planning problem can be formally stated as follows. Let $\mathcal{X}$ be the configuration space of the robot and let $\Sigma(\mathcal{X})$ denote the set of all continuous functions $[0,1] \to \mathcal{X}$. The initial configuration of the robot is $\mathbf{x}_{\text{init}} \in \mathcal{X}$. The path is required to end in a goal region $X_{\text{goal}} \subseteq \mathcal{X}$. The set of all allowed configurations of the robot is called the free configuration space and denoted $\mathcal{X}_{\text{free}}$. Typically, the free configurations are those that do not result in collision with obstacles, but the free-configuration set can also represent other holonomic constraints[1] on the path. The differential constraints on the path are represented by a predicate $D(\mathbf{x}, \mathbf{x}', \mathbf{x}'', \ldots)$ and can be used to enforce some degree of smoothness of the path for the robot, such as the bound on the path curvature and/or the rate of curvature. For example, in the case of $\mathcal{X} \subseteq \mathbb{R}^2$, the differential constraint may enforce the maximum curvature κ of the path using Frenet-Serret formula as follows:

$$D(\mathbf{x}, \mathbf{x}', \mathbf{x}'', \ldots) \Leftrightarrow \frac{\|\mathbf{x}' \times \mathbf{x}''\|}{\|\mathbf{x}'\|^3} \leq \kappa.$$

Further, let $J : \Sigma(\mathcal{X}) \to \mathbb{R}$ be the cost functional. Then, the optimal version of the path planning problem can be generally stated as follows:

Problem 1 (Optimal path planning)**.** Given a 5-tuple $(\mathcal{X}_{\text{free}}, \mathbf{x}_{\text{init}}, X_{\text{goal}}, D, J)$ find $\sigma^* =$

$$\begin{array}{lll} \underset{\sigma \in \Sigma(\mathcal{X})}{\operatorname{argmin}} & J(\sigma) & \\ \text{subj. to} & \sigma(0) = \mathbf{x}_{\text{init}} \text{ and } \sigma(1) \in X_{\text{goal}} & \\ & \sigma(\alpha) \in \mathcal{X}_{\text{free}} & \forall \alpha \in [0,1] \\ & D(\sigma(\alpha), \sigma'(\alpha), \sigma''(\alpha), \ldots) & \forall \alpha \in [0,1]. \end{array}$$

The problem of feasible and optimal path planning has been studied extensively in the past few decades. The complexity of this problem is well understood, and many practical algorithms have been developed. The relevant complexity and algorithmic results will be discussed in the following two sections.

[1] Holonomic constraint is a constraint that is expressible in the form $f(\mathbf{x}) = 0$, i.e., it depends only on the configuration of the robot and not on the derivatives of the robot's configuration.

2.1.1.1 Complexity

A significant body of literature is devoted to studying the complexity of motion planning problems. The following is a brief survey of some of the major results regarding the computational complexity of these problems.

The problem of finding an optimal path subject to holonomic and differential constraints as formulated in Problem 1 is known to be PSPACE-hard [Reif, 1979]. This means that it is at least as hard as solving any NP-complete problem and thus, assuming $\mathrm{P} \neq \mathrm{NP}$, there is no efficient (polynomial-time) algorithm able to solve all instances of the problem. Research attention has since been directed toward studying approximate methods or approaches to subsets of the general motion planning problem.

Initial research focused primarily on feasible (i.e., non-optimal) path planning for a holonomic robot model in polygonal/polyhedral environments. That is, the obstacles are assumed to be polygons/polyhedra and there are no differential constraints on the resulting path. Reif [1979] found that an obstacle-free path for a holonomic robot, whose footprint can be described as a single polyhedron, can be found in polynomial time in both 2-d and 3-d environments. Later, Schwartz and Sharir [1983a] demonstrated that feasible path planning for a robot consisting of a *fixed* number of freely linked and independently controlled polyhedra in a 3-d environment with polyhedral obstacles is polynomially solvable – however, the provided algorithm is polynomial in the size of the description of the environment but exponential in the number of degrees of freedom. On the other hand, when the robot consists of n freely-linked bodies moving in a 2-d/3-d environment and n is not fixed, then the problem of finding a collision-free path for such a robot becomes PSPACE-hard [Reif, 1979, Hopcroft et al., 1984]. Canny [1988] has shown that the problem of feasible path planning in a free space represented using polynomials is in PSPACE, which rendered the decision version of feasible path planning without differential constraints as a PSPACE-complete problem.

In the optimal motion planning problem formulation, the objective is to find the *shortest* obstacle-free path. It has been long known that a shortest path for a circular holonomic robot in a 2-d environment with polygonal obstacles can be found in polynomial time [Lozano-Pérez and Wesley, 1979, Storer and Reif, 1994]. More precisely, it can be computed in time $O(n^2)$, where n is the number of vertices of the polygonal obstacles [Overmars and Welzl, 1988]. This can be solved by constructing and searching the so-called visibility graph [Berg et al., 2008]. In contrast, Lazard, Reif, and Wang [1998] established that the problem of finding a shortest curvature-bounded path in a 2-d plane amidst polygonal obstacles (i.e., a path for a car-like robot) is NP-hard, which suggests that there is no polynomial-time algorithm for finding a shortest path for a car-like robot among polygonal obstacles. A related result is that the existence of a curvature constrained path in a polygonal environment can be decided in EXPTIME [Fortune and Wilfong, 1991].

A special case where a solution can be efficiently computed is the shortest curvature bounded path in an obstacle free environment. Dubins [1957] has shown that the shortest path having curvature bounded by κ between given two points p_1, p_2 and with prescribed tangents θ_1, θ_2 is a curve consisting of at most three segments, each one being either a circular arc segment or a straight line. Three decades later, Reeds and Shepp [1990] extended the method for a car that can move both forwards and backwards.

2.1.1.2 Solution Techniques

Since for many problems of interest in robotics, exact algorithms with practical computational complexity are unavailable [Lazard et al., 1998], one has to resort to more general, approximative methods. These methods generally do not find an exact solution, but attempt to find a satisfactory solution or a sequence of feasible solutions that converge to an optimal solution. The utility and performance of these approaches are typically quantified by the class of problems for which they are applicable as well as their guarantees for converging to an optimal solution. The approximative methods for path planning can be broadly divided in three main categories:

- *Variational methods* represent the path as a function parameterized by a finite-dimensional vector

and the optimal path is sought by optimizing over the vector parameter using non-linear continuous optimization techniques. These methods are attractive for their rapid convergence to *locally* optimal solutions; however, they typically lack the ability to find globally optimal solutions unless an appropriate initial guess in provided. For a detailed discussion on variational methods, see Section 2.1.3.

- *Graph-search methods* discretize the configuration space of the robot as a graph, where the vertices represent a finite collection of robot configurations and the edges represent transitions between vertices. The desired path is found by performing a search for a minimum-cost path in such a graph. Graph search methods are not prone to get stuck in local minima, however, they are limited to optimize only over a discrete set of paths, namely those that can be constructed from the atomic motion primitives in the graph. For a detailed discussion about graph search methods, see Section 2.1.4.

- *Incremental search methods* sample the configuration space and incrementally build a reachability graph (oftentimes a tree) that maintains a discrete set of reachable configurations and feasible transitions between them. Once the graph is large enough so that at least one node is in the goal region, the desired path is obtained by tracing the edges that lead to that node from the start configuration. In contrast to more basic graph search methods, sampling-based methods incrementally increase the size of the graph until a satisfactory solution is found within the graph. For a detailed discussion about incremental search methods, see Section 2.1.5.

Clearly, it is possible to exploit the advantages of each of these methods by combining them. For example, one can use a coarse graph search to obtain an initial guess for the variational method as reported, e.g., by Lamiraux et al. [2004] and Boyer and Lamiraux [2006]. A comparison of key properties of selected path planning methods is given in Table 2.1.

2.1.2 Trajectory Planning

The motion planning problems in dynamic environments or with dynamic constraints may be more suitably formulated in the trajectory planning framework, in which the solution of the problem is a trajectory, i.e., a time-parameterized function $\pi : [0, T] \to \mathcal{X}$ prescribing the evolution of the configuration of the robot in time.

Let $\Pi(\mathcal{X}, T)$ denote the set of all continuous functions $[0, T] \to \mathcal{X}$ and $\mathbf{x}_{\text{init}} \in \mathcal{X}$ be the initial configuration of the robot. The goal region is $X_{\text{goal}} \subseteq \mathcal{X}$. The set of all allowed configurations at time $t \in [0, T]$ is denoted as $\mathcal{X}_{\text{free}}(t)$ and used to encode holonomic constraints such as the requirement on the path to avoid collisions with static and, possibly, dynamic obstacles. The differential constraints on the trajectory are represented by a predicate $D(\mathbf{x}, \mathbf{x}', \mathbf{x}'', \ldots)$ and can be used to enforce dynamic constraints on the trajectory. Further, let $J : \Pi(\mathcal{X}, T) \to \mathbb{R}$ be the cost functional. Then, the optimal version of the trajectory planning problem can be generally stated as:

Problem 2 (Optimal trajectory planning)**.** Given a 6-tuple $(\mathcal{X}_{\text{free}}, \mathbf{x}_{\text{init}}, X_{\text{goal}}, D, J, T)$ find $\pi^* =$

$$
\begin{array}{lll}
\underset{\pi \in \Pi(\mathcal{X}, T)}{\operatorname{argmin}} & J(\pi) & \\
\text{subj. to} & \pi(0) = \mathbf{x}_{\text{init}} \text{ and } \pi(T) \in X_{\text{goal}} & \\
& \pi(t) \in \mathcal{X}_{\text{free}}(t) & \forall t \in [0, T] \\
& D(\pi(t), \pi'(t), \pi''(t), \ldots) & \forall t \in [0, T].
\end{array}
$$

2.1.2.1 Complexity

Since trajectory planning in a dynamic environment is a generalization of path planning in static environments, the problem remains PSPACE-hard. Moreover, trajectory planning in dynamic environments

	Model assumptions	Completeness	Optimality	Time Complexity	Anytime
Visibility graph [Lozano-Pérez and Wesley, 1979]	2-d polyg. conf. space, no diff. constraints	Yes [a]	Yes	$O(n^2)$ [b] [Overmars and Welzl, 1988]	No
Cyl. algebr. decomp. [Canny and Reif, 1987]	No diff. constraints	Yes	No	Exp. in dimension [Canny, 1988]	No
Variational methods (Sec 2.1.3)	Lipschitz-continuous Jacobian	No	Locally optimal	$O(1/\epsilon)$ [k,l] [Bertsekas, 1999]	Yes
Road lane graph + Dijkstra (Sec 2.1.4.1)	Arbitrary	No [c]	No [d]	$O(n + m \log m)$ [e,f] [Fredman and Tarjan, 1987]	No
Lattice/tree of motion prim. + Dijkstra (Sec 2.1.5)	Arbitrary	No [c]	No [d]	$O(n + m \log m)$ [e,f] [Fredman and Tarjan, 1987]	No
PRM + Dijkstra [g] [Kavraki et al., 1998]	Exact steering procedure available	Probabilistically complete * [Karaman and Frazzoli, 2011]	Asymptotically optimal * [Karaman and Frazzoli, 2011]	$O(n^2)$ [h,f,*] [Karaman and Frazzoli, 2011]	No
PRM* + Dijkstra [Karaman and Frazzoli, 2011, Schmerling et al., 2015]	Exact steering procedure available	Probabilistically complete *,† [Karaman and Frazzoli, 2011, 2013, Schmerling et al., 2015]	Asymptotically optimal *,† [Karaman and Frazzoli, 2011, 2013, Schmerling et al., 2015]	$O(n \log n)$ [h,f,*,†] [Karaman and Frazzoli, 2011, 2013]	No
RRG + Dijkstra [Karaman and Frazzoli, 2011]	Exact steering procedure available	Probabilistically complete * [Karaman and Frazzoli, 2011]	Asymptotically optimal * [Karaman and Frazzoli, 2011]	$O(n \log n)$ [h,f,*] [Karaman and Frazzoli, 2011]	Yes
RRT [LaValle and Kuffner, 2001]	Arbitrary	Probabilistically complete [i,*] [LaValle and Kuffner, 2001]	Suboptimal * [Karaman and Frazzoli, 2011]	$O(n \log n)$ [h,f,*] [Karaman and Frazzoli, 2011]	Yes
RRT* [Karaman and Frazzoli, 2011]	Exact steering procedure available	Probabilistically complete *,† [Karaman and Frazzoli, 2011, 2013]	Asymptotically optimal *,† [Karaman and Frazzoli, 2011, 2013]	$O(n \log n)$ [h,f,*,†] [Karaman and Frazzoli, 2011, 2013]	Yes
SST* [Li et al., 2015]	Lipschitz-continuous dynamics	Probabilistically complete † [Li et al., 2015]	Asymptotically optimal † [Li et al., 2015]	N/A [j]	Yes

Table 2.1: Comparison of path planning methods. **Legend:** a: for the shortest path problem; b: n is the number of points defining obstacles; c: complete only w.r.t. the set of paths induced by the given graph; d: optimal only w.r.t. the set of paths induced by the given graph; e: n and m are the number of edges and vertices in the graph respectively; f: assuming $O(1)$ collision checking; g: batch version with fixed-radius connection strategy; h: n is the number of samples/algorithm iterations; i: for certain variants; j: not explicitly analyzed; k: ϵ is the required distance from the optimal cost; l: faster rates possible with additional assumptions; $*$: shown for systems without differential constraints; †: shown for some class of nonholonomic systems.

has been shown to be harder than path planning in the sense that some variants of the problem that are tractable in static environments become intractable when an analogical problem is considered in a dynamic environment. In particular, recall that a shortest path for a point robot in a static 2-d polygonal environment can be found efficiently in polynomial time and contrast it with the result of Canny and Reif [1987] establishing that finding velocity-bounded collision-free trajectory for a holonomic point robot amidst moving polygonal obstacles[2] is NP-hard. Similarly, while path planning for a robot with a fixed number of degrees of freedom in 3-d polyhedral environments is tractable, Reif and Sharir [1994] established that trajectory planning for robot with 2 degrees of freedom among translating and rotating 3-d polyhedral obstacles is PSPACE-hard.

2.1.2.2 Solution Techniques

Since tractable exact algorithms are not available for most trajectory planning problems occurring in robotics, the numerical methods are a popular choice for the task. Trajectory planning problems can be numerically solved using most variational methods directly in the time domain or by converting the trajectory planning problem to path planning in a configuration space with an added time-dimension [Fraichard, 1998]:

The conversion from a trajectory planning problem $(\mathcal{X}^T_{\text{free}}, \mathbf{x}^T_{\text{init}}, X^T_{\text{goal}}, D^T, J^T, T)$ to a path planning problem $(\mathcal{X}^P_{\text{free}}, \mathbf{x}^P_{\text{init}}, X^P_{\text{goal}}, D^P, J^P)$ is usually done as follows. The configuration space where the path planning takes place is defined as $\mathcal{X}^P := \mathcal{X}^T \times [0, T]$. For any $\mathbf{y} \in \mathcal{X}^P$, let $t(\mathbf{y}) \in [0, T]$ denote the time component and $c(\mathbf{y}) \in \mathcal{X}^T$ denote the "configuration" component of the point $\mathbf{y}$. A path $\sigma : [0, 1] \to \mathcal{X}^P$ can be converted to a trajectory $\pi : [0, T] \to \mathcal{X}^T$ if the time component at start and end point of the path is constrained as

$$\begin{aligned} t(\sigma(0)) &= 0 \\ t(\sigma(1)) &= T \end{aligned}$$

and the path is monotonically-increasing, which can be enforced by a differential constraint

$$t(\sigma'(\alpha)) > 0 \qquad \forall \alpha \in [0, 1].$$

Further, the free configuration space, initial configuration, goal region and differential constraints is mapped to their path planning counterparts as follows:

$$\begin{aligned} \mathcal{X}^P_{\text{free}} &= \{(\mathbf{x}, t) : x \in \mathcal{X}^T_{\text{free}}(t) \wedge t \in [0, T]\} \\ \mathbf{x}^P_{\text{init}} &= (\mathbf{x}^T_{\text{init}}, 0) \\ X^P_{\text{goal}} &= \{(\mathbf{x}, T) : \mathbf{x} \in X^T_{\text{goal}}\} \\ D^P(\mathbf{y}, \mathbf{y}', \mathbf{y}'', \ldots) &= D^T(c(\mathbf{y}), \tfrac{c(\mathbf{y}')}{t(\mathbf{y}')}, \tfrac{c(\mathbf{y}'')}{t(\mathbf{y}'')}, \ldots). \end{aligned}$$

A solution to such a path planning problem is then found using a path planning algorithm that can handle differential constraints and converted back to the trajectory form. A specific algorithm based on a search in a discretized version of the space-time is formulated and analysed in Section 4.4.

Speed-up techniques Since the path planning in space-time can be computationally expensive, many authors try to tackle the complexity by decomposing the problem into a spatial part and time-dependent part. Some approaches simply plan an optimal spatial path first, and subsequently, search for a speed profile along the path that avoids collisions with moving obstacles [Kant and Zucker, 1986]. Van den Berg and Overmars [2005b] proposed a more general two-level technique that constraints the spatial motions to a precomputed spatial roadmap in the environment and performs space-time search for a

[2] In fact, the authors considered an even more constrained *2-d asteroid avoidance problem*, where the task is to find a collision-free trajectory for a point robot with a bounded velocity in a 2-d plane with convex polygonal obstacles moving with a fixed linear speed.

discretized collision-free speed profile only on the edges of the roadmap. Fujimura [1995] proposes a method for finding time-minimum routes in a route network amidst moving obstacles with piecewise linear motions. The point robot is constrained to move on the given route network, but it can choose any speed profile within given velocity bounds to move along the edges of the network. The author gives a complete polynomial algorithm that finds exact time-minimal trajectories on the given route network.

If the objective of the trajectory planning is to minimize the time of arrival to the goal region, then the safe-interval path planning (SIPP) technique [Phillips and Likhachev, 2011, Narayanan et al., 2012] can be used to significantly reduce the size of the state space that has to be searched. The safe-interval technique is based on the observation that the time axis for each spatial position can be divided into intervals when the position is in collision with a moving obstacle and intervals when it is safe. It can be shown that it is sufficient to only maintain one state for each safe interval at each spatial position, which dramatically reduces the number of states that have to be considered.

In the remainder of the section, we will discuss the individual approaches to single-robot path planning in more detail.

2.1.3 Variational Methods

We will first address the trajectory planning problem in the framework of nonlinear continuous optimization. In this context, the problem is often referred to as trajectory optimization. Within this subsection we will adopt the trajectory planning formulation with the understanding that doing so does not affect generality since path planning can be formulated as trajectory optimization over the unit time interval.

Most nonlinear programming techniques require the trajectory planning problem, as formulated in Problem 2, to be converted into the form

$$\begin{array}{ll} \underset{\pi\in\Pi(\mathcal{X},T)}{\operatorname{argmin}} & J(\pi) \\ \text{subj. to} & \pi(0)=\mathbf{x}_{\text{init}} \text{ and } \pi(T)\in X_{\text{goal}} \\ & f(\pi(t),\pi'(t),\ldots)=0 \qquad \forall t\in[0,T] \\ & g(\pi(t),\pi'(t),\ldots)\leq 0 \qquad \forall t\in[0,T], \end{array} \tag{2.1.1}$$

where the holonomic and differential constraints are represented by a system of equality and inequality constraints. As is common in the field of constrained optimization, the above problem is cast as an unconstrained optimization problem by replacing the equality and inequality constraints with penalty and barrier functions that augment the cost functional. With the penalty method, the cost functional takes the form

$$\tilde{J}(\pi)=J(\pi)+\frac{1}{\varepsilon}\int_0^T\left[\|f(\pi(t),\pi(t)',\ldots)\|^2+\|\max(0,g(\pi(t),\pi(t)',\ldots))\|^2\right]\mathrm{d}t.$$

Similarly, barrier functions can be used in place of inequality constraints. The augmented cost functional in this case takes the form

$$\tilde{J}(\pi)=J(\pi)+\varepsilon\int_0^T h(\pi(t),\pi(t)',\ldots)\mathrm{d}t\ ,$$

where $h(\mathbf{x},\mathbf{x}',\mathbf{x}'',\ldots)$ is the barrier function satisfying $g(\mathbf{x},\mathbf{x}',\mathbf{x}'',\ldots)<0\Rightarrow h(\mathbf{x},\mathbf{x}',\mathbf{x}'',\ldots)<\infty$, $g(\mathbf{x},\mathbf{x}',\mathbf{x}'',\ldots)\geq 0\Rightarrow h(\mathbf{x},\mathbf{x}',\mathbf{x}'',\ldots)=\infty$, and $\lim_{g(\mathbf{x},\mathbf{x}',\mathbf{x}'',\ldots)\to 0}\{h(\mathbf{x},\mathbf{x}',\mathbf{x}'',\ldots)\}=\infty$. The intuition behind both of the augmented cost functionals is that by making ε small, minima in cost will be close to minima of the original cost functional. An advantage of barrier functions is that local minima remain feasible, but must be initialized with a feasible solution to have finite augmented cost. Penalty methods, on the other hand, can be initialized with any trajectory and optimized to a local minimum. However, local minima may violate the problem constraints. A variational formulation using barrier functions is used e.g. by Rucco et al. [2012] for vehicle trajectory planning.

The trajectory optimization problems are generally nonlinear and thus typically do not have an analytic solution. Therefore, they have to be solved numerically. The two major solution approaches are indirect methods and direct methods:

Indirect methods exploit Pontryagin's minimum principle [Pontryagin, 1987], one of the central results in optimal control, which provides optimality conditions of a solution to a variational problem. These methods solve the problem by finding solutions satisfying these optimality conditions, which translates to a two-point boundary value problem that is typically, in turn, solved numerically.

Direct methods translate a variational problem to a finite-dimensional optimization problem by considering a finite dimensional subspace of $\Pi(\mathcal{X}, T)$. To this end, it is usually assumed that

$$\pi(t) \approx \tilde{\pi}(t) = \sum_{i=1}^{N} q_i \phi_i(t),$$

where q_i is a coefficient from $\mathbb{R}$, and $\phi_i(t)$ are *basis* functions. A number of numerical approximation schemes have proven useful for representing the trajectory optimization problem as a nonlinear program such as collocation [Betts, 2010] and pseudospectral methods [Ross and Karpenko, 2012]. The existing nonlinear optimization methods can be then used to solve the resulting finite-dimensional problem.

The topic of variational approaches is very extensive and hence, the above is only a brief overview of selected approaches. We refer the interested reader to [Betts, 1998, Polak, 1973] for dedicated surveys on this topic.

2.1.4 Graph Search Methods

Although useful in many contexts, the applicability of variational methods is limited by their convergence to only local minima. In this section, we will discuss the class of methods that attempts to mitigate the problem by performing a global search in a discretized version of the path space. These so-called graph search methods discretize the configuration space $\mathcal{X}$ of the vehicle and represent it in the form of a graph and then search for a minimum cost path on such a graph.

In this approach, the configuration space is represented as a graph $G = (V, E)$, where $V \subset \mathcal{X}$ is a discrete set of selected configurations called vertices and $E = \{(o_i, d_i, \sigma_i)\}$ is the set of edges, where $o_i \in V$ represents the origin of the edge, d_i represents the destination of the edge and σ_i represents the path segment connecting o_i and d_i. It is assumed that the path segment σ_i connects the two vertices, i.e., $\sigma_i(0) = o_i$ and $\sigma_i(1) = d_i$. Further, it is assumed that the initial configuration $\mathbf{x}_{\text{init}}$ is a vertex of the graph. The edges are constructed in such a way that the path segments associated with them lie completely in $\mathcal{X}_{\text{free}}$ and satisfy differential constraints. As a result, any path on the graph can be converted to a feasible path for the robot by concatenating the path segments associated with edges of the path through the graph.

There is a number of strategies for constructing a graph discretizing the free configuration space of a robot. In the following subsections, we discuss three common strategies: Hand-crafted lane graphs, graphs derived from geometric representations and graphs constructed by either control or configuration sampling.

2.1.4.1 Lane Graph

When the path planning problem involves driving in a structured environment, e.g., in a road network in the context of autonomous driving, a sufficient graph discretization may consist of edges representing the path that the robotic vehicle should follow within each lane and paths that traverse intersections.

Road lane graphs are often partly algorithmically generated from higher-level street network maps and partly human edited. An example of such a graph is in Figure 2.1.1.

Although most of the time it is sufficient for the autonomous vehicle to follow the paths encoded in the road lane graph, occasionally it must be able to navigate around obstacles that were not considered

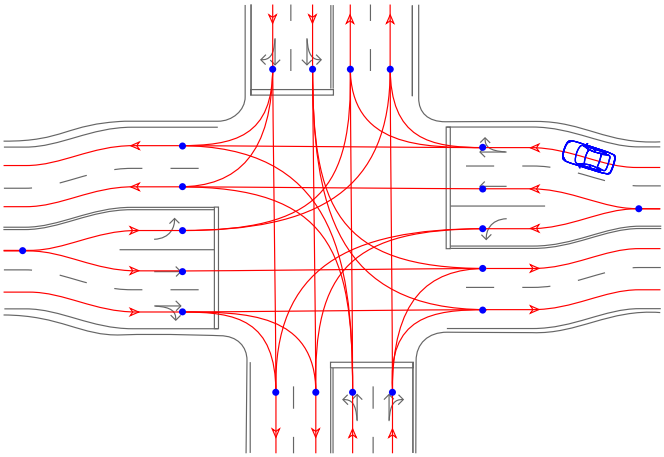


Figure 2.1.1: Hand-crafted graph representing desired driving paths under normal circumstances.

when the road network graph was designed or in environments not covered by the graph. Consider, for example, a faulty vehicle blocking the lane that the vehicle plans to traverse – in such a situation a more general motion planning approach is necessary to find a collision-free path around the detected obstacle.

The general path planning approaches can be broadly divided into two categories based on how they represent the obstacles in the environment. So-called geometric or combinatorial methods work with geometric representations of the obstacles, where in practice the obstacles are most commonly described using polygons or polyhedra. On the other hand, so-called sampling-based methods abstract away from how the obstacles are internally represented and only assumes access to a function that determines if any given path segment is in a collision with any of the obstacles.

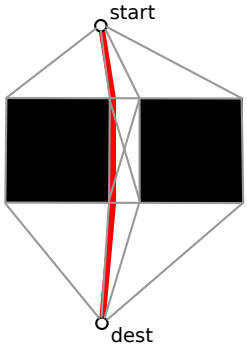
2.1.4.2 Geometric Methods

In this section, we will focus on path planning methods that work with geometric representations of obstacles.

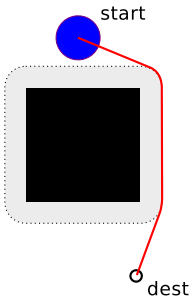
In path planning, the term roadmap is used to refer to a graph discretization of $\mathcal{X}_{\text{free}}$ that describes well the connectivity of the free configuration space and has the property that any point in $\mathcal{X}_{\text{free}}$ is trivially reachable from some vertex in the roadmap. When the set $\mathcal{X}_{\text{free}}$ can be described geometrically using a linear or semi-algebraic model, different types of roadmaps for $\mathcal{X}_{\text{free}}$ can be algorithmically constructed and subsequently used to obtain complete path planning algorithms. Most notably, for $\mathcal{X}_{\text{free}} \subseteq \mathbb{R}^2$ and polygonal models of the configuration space, several efficient algorithms for constructing such roadmaps exists such as the vertical cell decomposition [Chazelle, 1987], generalized Voronoi diagrams [Ó'Dúnlaing and Yap, 1985, Takahashi and Schilling, 1989], and visibility graphs [Latombe, 2012, Nilsson, 1969].

In the visibility graph approach [Berg et al., 2008, Lozano-Pérez and Wesley, 1979], the obstacle-free space is discretized in form of a graph and subsequently a shortest path on this graph is found using a graph search algorithm. The vertices of the visibility graph are corners of the polygons representing the obstacles, the start point, and the goal point. Two vertices in a visibility graph are connected by an edge if and only if a straight line between them does not intersect any of the obstacles. An important property of this approach is that a shortest path, if it exists, can be always constructed using edges from the visibility graph and thus this approach is guaranteed to find an optimal solution if it exists. An example of a visibility graph with the optimal path highlighted in red is in Figure 2.1.2a.

Observe that the problem of finding a shortest path for a circular robot amidst polygonal obstacles can be easily converted to the problem of finding the shortest path for a point robot amidst polygonal



(a) Computing the shortest path using the visibility graph. The task is to find the shortest path from `start` to `dest` such that the obstacles, depicted in black in the picture, are avoided. The visibility graph is shown in gray, the shortest path on the graph is highlighted in red.



(b) Finding a shortest path for a circular robot. The task is to find a shortest path for a circular robot in blue from the start position to the destination position around the black obstacle. The solution can be found by finding a shortest path for a point robot (the center of the circle) around the inflated obstacle depicted in gray.

Figure 2.1.2: Finding the shortest path using a visibility graph

obstacles by "inflating" the obstacles by the radius of the robot [Lozano-Pérez and Wesley, 1979]. Such a transformation exploits the fact that a circular body with radius r does not intersect any obstacle if and only if the center of the circular body is at at least at distance r from the nearest obstacle. The same transformation is also known as dilation in mathematical morphology or buffer in GIS terminology. The result of polygon inflation is a region bounded by lines and circular segments, which can be converted to a polygon by approximating the circular segments by a sequence of line segments. Polygonal approximations of inflated obstacles are then used to find a shortest path for the center of the circular body of the robot using the shortest path roadmap. For an illustration of the process see Figure 2.1.2b.

Unfortunately, the algorithms based on a search in a visibility graph do not generalize to higher-dimensional spaces, not even to planning in 3-d space. For higher dimensional configuration spaces described by a general semi-algebraic model, the technique known as cylindrical algebraic decomposition can be used to construct a roadmap in the configuration space [LaValle, 2006, Schwartz and Sharir, 1983b] leading to complete, but non-optimal, algorithms for a very general class of path planning problems. The fastest of this class is an algorithm developed by Canny [1988] that has (single) exponential time complexity in the dimension of the configuration space. The result is however mostly of a theoretical nature without any known implementation to date.

Due to its relevance to path planning for car-like vehicles, a number of results also exist for the problem of path planning with a constraint on maximum curvature. Backer and Kirkpatrick [2007] provide an algorithm for constructing a path with a bounded curvature that is polynomial in the number of features of the domain, the precision of the input and the number of segments on the simplest obstacle-free Dubins path connecting the specified configurations. Since the problem of finding a *shortest* path with a bounded curvature amidst polygonal obstacles is NP-hard, it is not surprising that no exact polynomial solution algorithm is known. An approximation algorithm for finding a shortest curvature-bounded path amidst polygonal obstacles has been first proposed by Jacobs and Canny [1993] and later improved by Wang and Agarwal [1996] with time complexity $O(\frac{n^2}{\epsilon^4}\log n)$, where n is the number of vertices of the obstacles and ϵ is the approximation factor.

2.1.4.3 Sampling-based Methods

Often, a geometric model of $\mathcal{X}_{\text{free}}$ is not directly available or it would be too costly to construct from raw sensory data. Moreover, the requirements on the resulting path are often far more complicated than

a simple maximum curvature constraint. This may explain the popularity of sampling-based techniques that do not enforce a specific representation of the free configuration set and dynamic constraints. Instead of reasoning over a geometric representation, the sampling based methods explore the reachability of the free configuration space using *steering* and *collision checking* routines:

The steering function `steer(`$\mathbf{x}, \mathbf{y}$`)` returns a path segment starting from configuration $\mathbf{x}$ going towards configuration $\mathbf{y}$ (but not necessarily reaching $\mathbf{y}$) ensuring the differential constraints are satisfied, i.e., the resulting motion is feasible for the robot model in consideration. The exact manner in which the steering function is implemented depends on the context in which it is used. Some typical choices encountered in the literature are:

1. Random steering: The function returns a path that results from applying a random control input through a forward model of the robot from state $\mathbf{x}$ for either a fixed or variable time step [La Valle, 1998].

2. Heuristic steering: The function returns a path that results from applying control that is heuristically constructed to guide the system from $\mathbf{x}$ towards $\mathbf{y}$ [Petti and Fraichard, 2005, Bhatia and Frazzoli, 2004, Glassman and Tedrake, 2010]. This includes selecting the maneuver from a predesigned discrete set (library) of maneuvers.

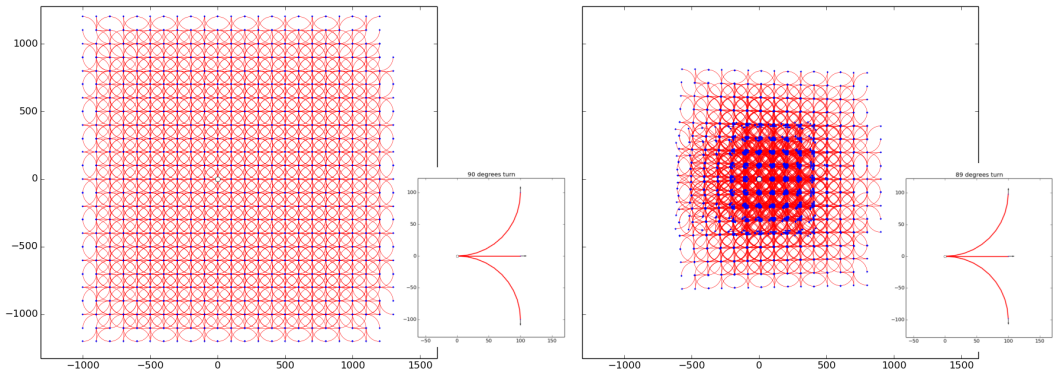
3. Exact steering: The function returns a feasible path that guides the system from $\mathbf{x}$ to $\mathbf{y}$. Such a path corresponds to a solution of a 2-point boundary value problem. For some systems and cost functionals, such a path can be obtained analytically, e.g., a straight line for holonomic systems, a Dubins curve for forward-moving unicycle [Dubins, 1957], or a Reeds-Shepp curve for bidirectional unicycle [Reeds and Shepp, 1990]. An analytic solution also exists for differentially flat systems [Jeon et al., 2013], while for more complicated models, the exact steering can be obtained by solving the corresponding two-point boundary value problem.

4. Optimal exact steering: The function returns an optimal exact steering path with respect to the given cost functional. In fact, the straight line, the Dubins curve, and the Reeds-Shepp curve from the previous point are optimal solutions assuming that the cost functional is the arc-length of the path [Dubins, 1957, Reeds and Shepp, 1990].

The collision checking function `col-free(`σ`)` returns true if path a given segment σ lies entirely in $\mathcal{X}_{\text{free}}$ and it is used to ensure that the resulting path does not collide with any of the obstacles.

Having access to steering and collision checking functions, the major challenge becomes how to construct a discretization that approximates well the connectivity of $\mathcal{X}_{\text{free}}$ without having access to an explicit model of its geometry. We will now review sampling-based discretization strategies from literature. A straightforward approach is to choose a set of motion primitives (fixed maneuvers) and generate the search graph by recursively applying them starting from the robot's initial configuration $\mathbf{x}_{\text{init}}$, e.g., using the method in Algorithm 1. For path planning without differential constraints, the motion primitives can be simply a set of straight lines with different directions and lengths. For a car-like robot, the motion primitives might be a set of arcs representing the path the car would follow with different values of steering. A variety of techniques can be used for generating motion primitives for a particular robotic system. A common approach is to sample a number of control inputs and to simulate forwards in time using a robot model to obtain feasible motions.

Observe that the recursive application of motion primitives may generate a tree graph in which, in the worst-case, no two edges lead to the same configuration. There are, however, sets of motion primitives, referred to as lattice-generating, that result in regular graphs resembling a lattice. See Figure 2.1.3a for an illustration. The advantage of lattice generating primitives is that the vertices of the search graph cover the configuration space uniformly, while trees in general may have a high density of vertices around the root vertex. Pivtoraiko et al. use the term "state lattice" to describe such graphs in [Pivtoraiko et al., 2009] and point out that a set of lattice-generating motion primitives for a system in hand can be obtained by first generating regularly spaced configurations around origin and then connecting the origin



(a) Lattice graph (b) Non-Lattice graph

Figure 2.1.3: Lattice and non-lattice graph, both with 5000 edges. (a) The graph resulting from recursive application of 90° left circular arc, 90° right circular arc, and a straight line. (b) The graph resulting from recursive application of 89° left circular arc, 89° right circular arc, and a straight line. The recursive application of those primitives does form a tree instead of a lattice with many branches looping in the neighborhood of the origin. As a consequence, the area covered by the right graph is smaller.

to such configurations by a path that represents the solution to the two-point boundary value problem between the two configurations.

Algorithm 1: Recursive Roadmap Construction

```
1  V ← {x_init}; E ← ∅; Q ← new queue(x_init);
2  while Q ≠ ∅ do
3      x ← pop element from Q;
4      M ← generate a set of path segments by applying motion primitives from configuration x;
5      for σ ∈ M do
6          if col-free(σ) then
7              E ← E ∪ {(x, σ(1), σ)};
8              if σ(1) ∉ V then
9                  add σ(1) to Q;
10                 V ← V ∪ {σ(1)};
11 return (V, E)
```

An effect that is similar to recursive application of lattice-generating motion primitives from the initial configuration can be achieved by generating a discrete set of samples covering the (free) configuration space and connecting them by feasible path segments obtained using an exact steering procedure.

Most sampling-based roadmap construction approaches follow the algorithmic scheme shown in Algorithm 2, but differ in the implementation of the `sample-points`$(\mathcal{X}, n)$ and `neighbors`$(\mathbf{x}, V)$ routines. The function `sample-points`$(\mathcal{X}, n)$ represents the strategy for selecting n points from the configuration space $\mathcal{X}$, while the function `neighbors`$(\mathbf{x}, V)$ represents the strategy for selecting a set of neighboring vertices $N \subseteq V$ for a vertex $\mathbf{x}$, which the algorithm will attempt to connect to $\mathbf{x}$ by a path segment using an exact steering function, $\texttt{steer}_{\text{exact}}(\mathbf{x}, \mathbf{y})$.

The two most common implementations of `sample-points`$(\mathcal{X}, n)$ function are 1) return n points arranged in a regular grid and 2) return n randomly sampled points from $\mathcal{X}$. While random sam-

Algorithm 2: Sampling-based Roadmap Construction

```
1 V ← {x_init} ∪ sample-points(X, n); E ← ∅;
2 for x ∈ V do
3     for y ∈ neighbors(x, V) do
4         σ ← steer_exact(x, y);
5         if col-free(σ) then
6             E ← E ∪ {(x, y, σ)};
7 return (V,E)
```

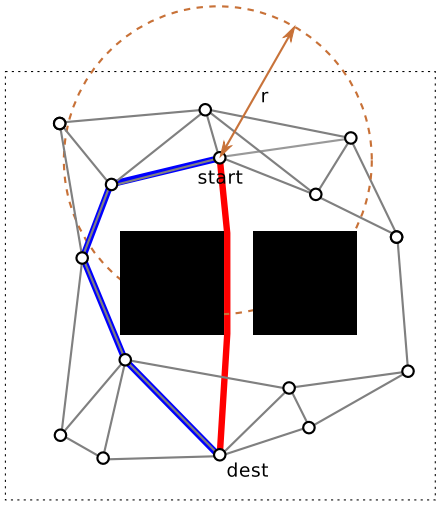


Figure 2.1.4: Example probabilistic roadmap with 15 vertices. The task is to find the shortest path from `start` to `dest` that does not intersect obstacles depicted in black. The optimal path is shown in red. The roadmap is shown in gray, the shortest path in the roadmap discretization is shown in blue.

pling has an advantage of being generally applicable and easy to implement, so-called Sukharev grids have been shown to achieve optimal L_∞-dispersion in unit hypercubes, i.e., they minimize the radius of the largest empty ball with no sample point inside. For in depth discussion of relative merits of random and deterministic sampling in the context of sampling-based path planning, we refer the reader to [LaValle et al., 2004]. The two most commonly used strategies for implementing `neighbors`$(\mathbf{x}, V)$ function are to take 1) the set of k-nearest neighbors to $\mathbf{x}$ or 2) the set of points lying within the ball centered at $\mathbf{x}$ with radius r.

In particular, samples arranged deterministically in a d-dimensional grid with the neighborhood taken as 4 or 8 nearest neighbors in 2-d or the analogous pattern in higher dimensions represents a straightforward deterministic discretization of the free configuration space. This is in part because they arise naturally from widely used bitmap representations of free and occupied regions of robots' configuration space [Lengyel et al., 1990].

Kavraki et al. [1996] advocate the use of random sampling within the framework of Probabilistic Roadmaps (PRM) in order to construct roadmaps in high-dimensional configuration spaces, because unlike grids, they can be naturally run in an anytime fashion. The batch version of PRM [Kavraki et al., 1998] follows the scheme in Algorithm 2 with random sampling and neighbors selected within a ball with fixed radius r. See Figure 2.1.4 for an example of a probability roadmap for a holonomic point robot in 2-d environment. Due to the general formulation of PRMs, they have been used for path planning for a variety of systems, including systems with differential constraints. However, the theoretical

analyses of the algorithm have primarily been focused on the performance of the algorithm for systems without differential constraints, i.e., when a straight line is used to connect two configurations. Under such an assumption, PRMs have been shown in [Karaman and Frazzoli, 2011] to be probabilistically complete and asymptotically optimal. That is, the probability that the resulting graph contains a valid solution (if it exists) converges to one with increasing size of the graph and the cost of the shortest path in the graph converges to the optimal cost. Karaman and Frazzoli [2011] further proposed an adaptation of batch PRM, called PRM*, that instead only connects neighboring vertices in a ball with a logarithmically shrinking radius with increasing number of samples to maintain both asymptotic optimality and computational efficiency.

In the same paper, the authors propose Rapidly-exploring Random Graphs (RRG*), which is an incremental discretization strategy that can be terminated at any time while maintaining the asymptotic optimality property. Recently, Fast Marching Tree (FMT*) [Janson et al., 2015] has been proposed as an asymptotically optimal alternative to PRM*. The algorithm combines discretization and search into one process by performing a lazy dynamic programming recursion over a set of sampled vertices that can be subsequently used to quickly determine the path from the initial configuration to the goal region.

The theoretical analysis has been extended also to differentially-constrained systems. In particular, Schmerling et al. [2015] proposed differential versions of PRM* and FMT* and proved asymptotic optimality of the algorithms for driftless control-affine dynamical systems, a class that includes models of non-slipping wheeled vehicles.

2.1.4.4 Graph Search Strategies

In the previous section, we have discussed techniques for the discretization of the free configuration space in the form of a graph. To obtain an actual optimal path in such a discretization, one must employ one of the graph search algorithms. In this section, we are going to review the graph search algorithms that are relevant for path planning.

The most widely recognized algorithm for finding shortest paths in a graph is probably the Dijkstra's algorithm [Dijkstra, 1959]. The algorithm performs the best first search to build a tree representing shortest paths from a given source vertex to all other vertices in the graph. When only a path to a single vertex is required, a heuristic can be used to guide the search process. The most prominent heuristic search algorithm is A* developed by [Hart, Nilsson, and Raphael, 1968]. If the provided heuristic function is admissible (i.e., it never overestimates the cost-to-go), A* has been shown to be optimally efficient and is guaranteed to return an optimal solution. For many problems, a bounded suboptimal solution can be obtained with less computational effort using Weighted A* [Pohl, 1970], which corresponds to simply multiplying the heuristic by a constant factor $\epsilon > 1$. It can be shown that the solution path returned by A* with such an inflated heuristics is guaranteed to be no worse than $(1 + \epsilon)$ times the cost of an optimal path.

Often, the shortest path from the robot's current configuration to the goal region is sought repeatedly every time the model of the world is updated using sensory data. Since each such update usually affects only a minor part of the graph, it might be wasteful to run the search every time completely from scratch. The family of real-time replanning search algorithms such as D* [Stentz, 1994], Focussed D* [Anthony, 1995] and D* Lite [Koenig and Likhachev, 2005] has been designed to efficiently recompute the shortest path every time the underlying graph changes, while making use of the information from previous search efforts.

Anytime search algorithms attempt to provide a first suboptimal path quickly and continually improve the solution with more computational time. Anytime A* [Hansen and Zhou, 2007] uses a weighted heuristic to find the first solution and achieves the anytime behavior by continuing the search with the cost of the first path as an upper bound and the admissible heuristic as a lower bound, whereas Anytime Repairing A* (ARA*) [Likhachev et al., 2003] performs a series of searches with inflated heuristic with decreasing weight and reuses information from previous iterations. On the other hand, Anytime Dynamic A* (ADA*) [Likhachev et al., 2005] combines ideas behind D* Lite and ARA* to produce an

anytime search algorithm for real-time replanning in dynamic environments.

A clear limitation of algorithms that search for a path on a graph discretization of the configuration space is that the resulting optimal path on such graph may be significantly longer than the true shortest path in the configuration space. Any-angle path planning algorithms [Daniel et al., 2010, Nash et al., 2010, Yap et al., 2011] are designed to operate on grids, or more generally on graphs representing cell decomposition of the free configuration space, and try to mitigate this shortcoming by considering "shortcuts" between the vertices on the graph during the search. In addition, Field D* introduces linear interpolation to the search procedure to produce smooth paths [Ferguson and Stentz, 2006].

2.1.5 Incremental Search Techniques

A disadvantage of the techniques that search over a fixed graph discretization is that they search only over the set of paths that can be constructed from primitives in the graph discretization. Therefore, these techniques may fail to return a feasible path or return a noticeably suboptimal one.

Incremental *feasible* motion planners strive to address this problem and provide a feasible path to any motion planning problem instance, if one exists, given enough computation time. Typically, these methods incrementally build increasingly finer discretization of the configuration space while concurrently attempting to determine if a path from the initial configuration to the goal region exists in the discretization at each step. If the instance is “easy”, the solution is provided quickly, but in general, the computation time can be unbounded. Similarly, incremental *optimal* motion planning approaches on top of finding a feasible path fast attempt to provide a sequence of solutions of increasing quality that converges to an optimal path.

The term *probabilistically complete* is used in the literature to describe algorithms that find a solution, if one exists, with probability approaching one with increasing computation time. Note that probabilistically complete algorithm may not terminate if the solution does not exist. Similarly, the term *asymptotically optimal* is used for algorithms that converge to an optimal solution with probability one.

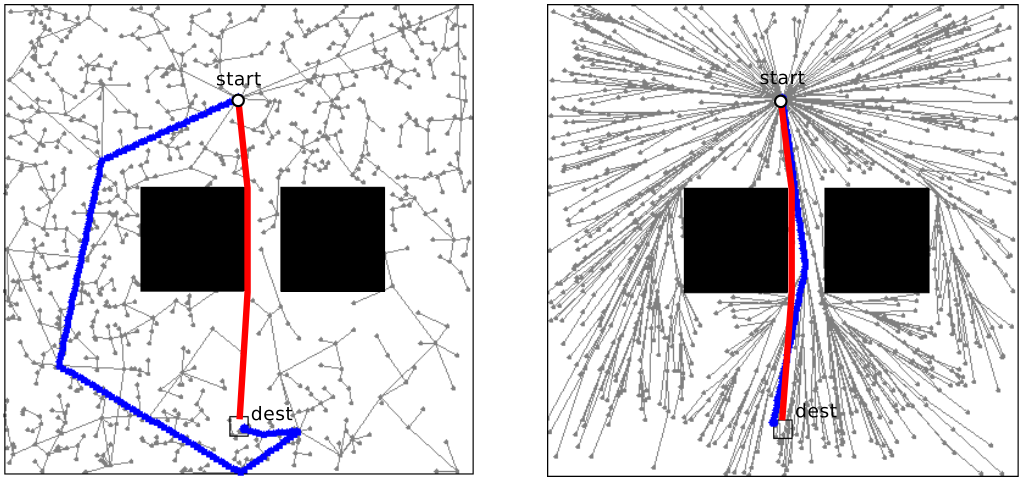
A naive strategy for obtaining completeness and optimality in the limit is to solve a sequence of path planning problems on a fixed discretization of the configuration space, each time with a higher resolution of the discretization. One disadvantage of this approach is that the path planning processes on individual resolution levels are independent without any information reuse. Moreover, it is not obvious how fast the resolution of the discretization should be increased before a new graph search is initiated, i.e., if it is more appropriate to add a single new configuration, double the number of configuration, or double the number of discrete values along each configuration space dimension. To overcome such issues, incremental motion planning methods interweave incremental discretization of configuration space with a search for a path within one integrated process.

An important class of methods for incremental path planning is based on the idea of incrementally growing a tree rooted at the initial configuration of the robot outwards to explore the reachable configuration space. The "exploratory" behavior is achieved by iteratively selecting a random vertex from the tree and by expanding the selected vertex by applying the steering function from it. Once the tree grows large enough to reach the goal region, the resulting path is recovered by tracing the links from the vertex in the goal region backwards to the initial configuration. The general algorithmic scheme of an incremental tree-based algorithm is described in Algorithm 3.

One of the first randomized tree-based incremental planners was the expansive spaces tree (EST) planner proposed by Hsu et al. [1997]. The algorithm selects a vertex for expansion, $\mathbf{x}_{\mathrm{selected}}$, randomly from V with a probability that is inversely proportional to the number of vertices in its neighborhood, which promotes growth towards unexplored regions. During expansion, the algorithm samples a new vertex $\mathbf{y}$ within a neighborhood of a fixed radius around $\mathbf{x}_{\mathrm{selected}}$, and use the same technique for biasing the sampling procedure to select a vertex from the region that is relatively less explored. Then it returns a straight line path between $\mathbf{x}_{\mathrm{selected}}$ and $\mathbf{y}$. A generalization of the idea for planning with kinodynamic constraints in dynamic environments was introduced in Hsu et al. [2002], where the capabilities of the algorithm were demonstrated on different non-holonomic robotic systems and the authors use an

Algorithm 3: Incremental Tree-based Algorithm

$V \leftarrow \{\mathbf{x}_{\text{init}}\}$; $E \leftarrow \emptyset$;
while *not interrupted* **do**
  $\mathbf{x}_{\text{selected}} \leftarrow$ `select(`V`);`
  $\sigma \leftarrow$ `extend(`$\mathbf{x}_{\text{selected}}, V$`);`
  if `col-free(`σ`)` **then**
    $\mathbf{x}_{\text{new}} = \sigma(1)$;
    $V \leftarrow V \cup \{\mathbf{x}_{\text{new}}\}$; $E \leftarrow E \cup \{(\mathbf{x}_{\text{selected}}, \mathbf{x}_{\text{new}}, \sigma)\}$;
return (V,E)



(a) RRT (b) RRT*

Figure 2.1.5: The tree generated by RRT (left) and RRT* (right) after 1000 algorithm iterations. The task is to find the optimal trajectory from `start` to goal region denoted as `dest` that avoids obstacles shown in black. The path found by each of the algorithms is shown in blue, the optimal path is shown in red.

idealized version of the algorithm to establish that the probability of failure to find a feasible path depends on the expansiveness property of the state space and decays exponentially with the number of samples.

Rapidly-exploring Random Trees (RRT) have been proposed by La Valle [1998] as an efficient method for finding feasible trajectories for high-dimensional non-holonomic systems. The rapid exploration is achieved by taking a random sample $\mathbf{x}_{\text{rnd}}$ from the free configuration space and extending the tree in the direction of the random sample. In RRT, the vertex selection function `select(`V`)` returns the nearest neighbor to the random sample $\mathbf{x}_{\text{rnd}}$ according to the given distance metric between the two configurations. The extension function `extend()` then generates a path in the configuration space by applying a control for a fixed time step that minimizes the distance to $\mathbf{x}_{\text{rnd}}$. Under certain simplifying assumptions (random steering is used for extension), the RRT algorithm has been shown to be probabilistic complete [LaValle and Kuffner, 2001]. We remark that the result on probabilistic completeness does not readily generalize to many practically implemented versions of RRT that often use heuristic steering. In fact, it has been recently shown in [Kunz and Stilman, 2015] that RRT using heuristic steering with fixed time step is not probabilistically complete.

Moreover, Karaman and Frazzoli [2011] demonstrated that the RRT converges to a suboptimal so-

lution with probability one and designed an asymptotically optimal adaptation of the RRT algorithm, called RRT*. As shown in Algorithm 4, the RRT* at every iteration considers a set of vertices that lie in the neighborhood of newly added vertex $\mathbf{x}_{\text{new}}$ and a) connects $\mathbf{x}_{\text{new}}$ to the vertex in the neighborhood that minimizes the cost of path from $\mathbf{x}_{\text{init}}$ to $\mathbf{x}_{\text{new}}$ and b) rewires any vertex in the neighborhood to $\mathbf{x}_{\text{new}}$ if that results in a lower cost path from $\mathbf{x}_{\text{init}}$ to that vertex. An important characteristic of the algorithm is that the neighborhood region is defined as the ball centered at $\mathbf{x}_{\text{new}}$ with radius being function of the size of the tree: $r = \gamma \sqrt[d]{(\log n)/n}$, where n is the number of vertices in the tree, d is the dimension of the configuration space, and γ is an instance-dependent constant. It is shown that for such a function, the expected number of vertices in the ball is logarithmic in the size of the tree, which is necessary to ensure that the algorithm almost surely converges to an optimal path while maintaining the same asymptotic complexity as the suboptimal RRT. See Figure 2.1.5 for a visual comparison of trees generated by RRT and RRT* after 1000 iterations.

Algorithm 4: RRT* Algorithm. The cost-to-come to vertex $\mathbf{x}$ is denoted as $c(\mathbf{x})$, the cost of path segment σ is denoted as $c(\sigma)$ and the parent vertex of vertex $\mathbf{x}$ is denoted by $p(\mathbf{x})$.

$V \leftarrow \{\mathbf{x}_{\text{init}}\}$; $E \leftarrow \emptyset$;
while *not interrupted* **do**
  $\mathbf{x}_{\text{selected}} \leftarrow$ `select(`V`);`
  $\sigma \leftarrow$ `extend(`$\mathbf{x}_{\text{selected}}, V$`);`
  if `col-free(`σ`)` **then**
    $\mathbf{x}_{\text{new}} = \sigma(1)$;
    $V \leftarrow V \cup \{\mathbf{x}_{\text{new}}\}$;
    `// consider all vertices in ball of radius` r `around` $\mathbf{x}_{\text{new}}$
    $r = \gamma \sqrt[d]{\frac{\log |V|}{|V|}}$;
    $X_{\text{near}} \leftarrow \{\mathbf{x} \in V \setminus \{\mathbf{x}_{\text{new}}\} : d(\mathbf{x}_{\text{new}}, \mathbf{x}\} < r\}$;
    `// find best parent`
    $\mathbf{x}_{\text{par}} \leftarrow \text{argmin}_{\mathbf{x} \in X_{\text{near}}}\ c(\mathbf{x}) + c($`connect(`$\mathbf{x}, \mathbf{x}_{\text{new}}$`)`$)$ subj. to
      `col-free(steer`$_{\text{exact}}(\mathbf{x}, \mathbf{x}_{\text{new}})$`)` ;
    $\sigma' =$ `steer`$_{\text{exact}}(\mathbf{x}_{\text{par}}, \mathbf{x}_{\text{new}})$;
    $E \leftarrow E \cup \{(\mathbf{x}_{\text{par}}, \mathbf{x}_{\text{new}}, \sigma')\}$;
    `// rewire vertices in neighborhood`
    for $\mathbf{x} \in X_{\text{near}}$ **do**
      $\sigma' \leftarrow$ `steer`$_{\text{exact}}(\mathbf{x}_{\text{new}}, \mathbf{x})$;
      if $c(\mathbf{x}_{\text{new}}) + c(\sigma') < c(\mathbf{x})$ *and* `col-free(`σ'`)` **then**
        $E \leftarrow (E \setminus \{(p(\mathbf{x}), \mathbf{x}, \sigma'')\}) \cup \{(\mathbf{x}_{\text{new}}, \mathbf{x}, \sigma')\}$,
          where σ'' is the path from $p(\mathbf{x})$ to $\mathbf{x}$;
return (V,E)

Sufficient conditions for asymptotic optimality of RRT* under differential constraints are stated in [Karaman and Frazzoli, 2011] and demonstrated to be satisfiable for Dubins vehicle and double integrator systems. In a later work, the authors further show in the context of small-time locally attainable systems that the algorithm can be adapted to maintain not only asymptotic optimality, but also computational efficiency [Karaman and Frazzoli, 2013]. Other related works focus on deriving distance and steering functions for non-holonomic systems by locally linearizing the system dynamics [Perez et al., 2012] or by deriving a closed-form solution for systems with linear dynamics [Webb and van den Berg,

2013]. On the other hand, RRT^{X} is an algorithm that extends RRT* to allow for real-time incremental replanning when the obstacle region changes, e.g., in the face of new data from sensors [Otte and Frazzoli, 2014].

New developments in the field of sampling-based algorithms include algorithms that achieve asymptotic optimality without having access to an exact steering procedure. In particular, Li et al. [2015] recently proposed the Stable Sparse Tree (SST) method for asymptotically (near-)optimal path planning, which is based on building a tree of randomly sampled controls propagated through a forward model of the dynamics of the system such that the locally suboptimal branches are pruned out to ensure that the tree remains sparse.

2.2 Multi-robot Coordination

In systems, where multiple robots operate in a shared area, one has to ensure that the robots do not collide not only with known static and dynamic obstacles but also with each other. The collision avoidance in multi-robot systems can be approached either within the reactive paradigm or within the deliberate paradigm:

- *Reactive approach* starts by computing an individually-optimal path for each robot to its destination without considering interactions with other robots. Then, the robots start following the planned paths while monitoring the positions and velocities of other robots in their surroundings. Whenever a potential future collision is detected, the robots attempt to avoid the collision by adjusting their current velocity vector. These methods are computationally efficient but susceptible to deadlocks. The reactive approaches are discussed in detail in Section 2.2.1.

- *Deliberative approach* is based on planning a coordinated motion for all robots prior execution. When each robot executes its assigned motion plan precisely (or within some tolerance that has been accounted for during planning), it is guaranteed that all robots reach their destinations without mutual collisions. The main challenge in the deliberative approach is how to efficiently compute the coordinated motions for the multi-robot team. The two major approaches to modeling and solving multi-robot motion planning problems are multi-robot motion planning and pebble motion problem:

  Multi-robot motion planning: When inter-robot collisions are modeled geometrically, then the problem of motion coordination can be stated as a generalization of the single-robot motion planning problem. Analogically to single-robot motion planning, multi-robot motion planning can be formulated both in the path planning framework and in the trajectory planning framework. The problem can be solved by translation to single-robot motion planning or with specialized algorithms that attempt to decouple the motion planning from conflict resolution. The former *coupled approach* to multi-robot motion planning always computes a solution (if one exists), but they may, in general, require computation time exponential in the number of robots. The latter *decoupled approach* is computationally more efficient but may, in general, fail to provide a solution. The multi-robot motion planning is discussed in more detail in Section 2.2.2.

  Pebble motion: The coordinated motions can be computed in an abstracted graph-based formulation, often referred to as pebble motion problem, where the geometry of the robots is not explicitly considered. Instead, the robots are assumed to move in discrete steps between vertices of a graph and a collision is defined as a situation when two robots occupy one vertex. In this formulation, suboptimal solutions can be obtained in polynomial time, but the formalism may not be expressive enough to encode geometric and dynamic constraints appearing in many practical applications. The pebble motion problem and the related solution algorithms are discussed in Section 2.2.3.

The main properties of different approaches to multi-robot coordination are summarized in Table 2.2.

	Model Assumptions	Guaranteed solution?	Complexity	Solution quality	Decentralization
Reactive techniques	General robot model [Bareiss and van den Berg, 2015]	No, deadlocks may occur	P[1]; demonstrated on thousands of agents [van den Berg et al., 2011]	Acceptable in open space, poor in clutter	Suited, requires ability to observe position and velocity of neighbor robots
Coupled methods	General robot model [Wagner and Choset, 2015]	Yes[2]	NP-hard, $O(c^n)$ [4] [Spirakis and Yap, 1984]	Optimal[2]	Not suited
Decoupled methods	General robot model	No	-; Demonstrated on tens of robots	Suboptimal, acceptable	Suited, requires communication [Velagapudi et al., 2010, Čáp et al., 2015b]
Pebble motion	No geometric constraints, no differential constraints, discrete moves	Yes	Non-optimal: P [Kornhauser et al., 1984] Optimal: NP-Hard [Yu, 2016]	Non-optimal version: Poor, sequential solution	Not suited
Proposed (RPP, AD-RPP, COBRA)	General robot model[3]	Yes[2] (in well-formed infrastructures) [Čáp et al., 2015b]	P (in well-formed infrastructures) [Čáp et al., 2015b]	Suboptimal, acceptable	Suited, requires communication [Čáp et al., 2015b]

Table 2.2: Summary of main multi-robot coordination techniques . **Notes:** [1]computing collision avoiding velocity in one iteration after linear relaxation; [2] with respect to a chosen discretization; [3] formal analysis assumes holonomic robots, practically demonstrated on non-holonomic systems [Čáp et al., 2015b, Čáp et al., 2013]; [4] n is the number of robots, c is a constant

2.2.1 Reactive Methods

In the reactive approach, the robot follows the shortest path (or other desired path) to its current destination and attempts to resolve collision situations locally as they appear. Each robot periodically observes positions and velocities of other robots in its neighborhood. If there is a potential future collision, the robot attempts to avert the collision by adjusting its immediate heading and velocity. Many strategies prescribing how to choose a collision-avoiding velocity have been proposed within the last three decades. Among the first ones has been the cocktail party model proposed by Lumelsky and Harinarayan [1997], while more recently, the techniques based on the velocity obstacle paradigm [Fiorini and Shiller, 1998] have become dominant [van den Berg et al., 2008, Guy et al., 2009, van den Berg et al., 2011]. The latter collision avoidance techniques are executed in a closed loop by each robot independently. At each iteration, a particular robot r observes other robots in its neighborhood and constructs a velocity obstacle for each observed robot q. The velocity obstacle of robot q to robot r is a set of velocities of robot r that would lead to a collision with the robot q assuming the robot q will maintain its current velocity. Then, the robot r finds the closest velocity vector to its desired velocity vector that is outside velocity obstacles that correspond to the robots in its neighborhood and follows the computed velocity vector in the next time step. When implemented naively, oscillations can occur but can be avoided by dividing the collision avoidance effort among the robots [van den Berg et al., 2008]. A velocity obstacle of a single robot is generally a cone-shaped region and the union of velocity obstacles of multiple robots can be a complicated non-convex region in the velocity space. In ClearPath algorithm [Guy et al., 2009], the optimal velocity vector is computed using quadratic programming with non-convex constraints representing the collision regions, while in the optimal reciprocal collision avoidance (ORCA) formulation [van den Berg et al., 2011], the optimal velocities are computed efficiently using linear programming thanks to a relaxation strategy that approximates each velocity obstacle by a half-plane in velocity space. While early work focused on collision-avoidance for holonomic robots, generalizations of the velocity obstacle paradigm to multi-robot systems with non-holonomic robots also exist [Alonso-Mora et al., 2013, Bareiss and van den Berg, 2015].

The approaches based on reciprocal velocity obstacles are widely used in practice thanks to their computational efficiency – a collision-avoiding velocity for a robot or other type of agent can be computed in a fraction of a millisecond [van den Berg et al., 2011], which makes them applicable to systems with a large number of agents such as robotic swarms and crowd simulation in computer games. However, the main limitation of the reactive techniques is that they resolve collisions only locally and thus cannot guarantee that the resulting motion will be deadlock-free and consequently that the robot will always reach its destination. See Figure 2.2.1 for an example scenario where robots end up in a deadlock while following a reactive collision avoidance strategy.

2.2.2 Multi-robot Motion Planning

Multi-robot path planning generalizes single-robot path planning by considering n articulated robots operating in a d-dimensional workspace$\mathcal{W} \subseteq \mathbb{R}^d$ that are required to avoid collisions not only with obstacles but also with each other. The pose of each robot i is completely described by its configuration, with the configuration space of robot i being denoted as $\mathcal{X}_i$. The region of workspace occupied by robot i when in configuration $\mathbf{x} \in \mathcal{X}_i$ is prescribed by a function $R_i : \mathcal{X}_i \to \mathcal{P}(\mathbb{R}^d)$, where $\mathcal{P}(S)$ denotes the power set of set S.

The multi-robot path planning problem is to find n paths $\sigma_i : [0,1] \to \mathcal{X}_i$, one for each robot $i = 1, \ldots, n$. The obstacle-free region of configuration space of robot i is $\mathcal{X}_i^{\text{free}} := \{\mathbf{x} : \mathbf{x} \in \mathcal{X}_i \text{ and } R_i(\mathbf{x}) \subset \mathcal{W}\}$. The initial configuration of robot i is denoted $\mathbf{x}_i^{\text{init}}$, the goal region of robot i is denoted X_i^{goal}. The differential constraints imposed on the path of robot i are represented by a predicate $D_i(\mathbf{x}, \mathbf{x}', \mathbf{x}'', \ldots)$. Further, let $J : \Sigma(\mathcal{X}_1) \times \ldots \times \Sigma(\mathcal{X}_n) \to \mathbb{R}$, where $\Sigma(\mathcal{X})$ denotes the set of all paths in $\mathcal{X}$, be a cost functional. The multi-robot path finding problem can be then formally defined as follows:

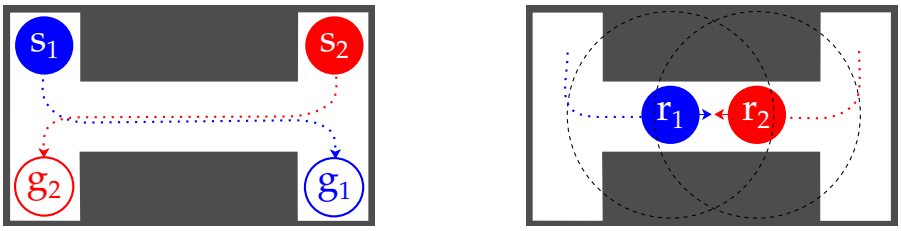


Figure 2.2.1: **Reactive methods and deadlocks:** Left: In this scenario, the robot 1 travels from s_1 to g_1 and the robot 2 travels from s_2 to g_2 in a corridor that is only slightly wider than a body of a single robot while using a reactive collision avoidance method based on the velocity obstacle paradigm. Right: The robots follow the shortest path to their respective destination while observing other robots in the neighborhood (depicted by a dashed circle). Eventually, the two robots meet within their collision detection distance and each of them attempts to compute a collision avoiding velocity. Here, the closest collision-avoiding velocity vector to the desired velocity vector (depicted as an arrow in the right figure) is the zero velocity vector and thus both robots stop and end up in a deadlock in the middle of the corridor.

Problem 3 (multi-robot path planning)**.** Given n quadruples $(\mathcal{X}_i^{\text{free}}, \mathbf{x}_i^{\text{init}}, X_i^{\text{goal}}, D_i)_{i=1,\dots n}$ and a cost functional J, find $\overline{\sigma}^* =$

$$\begin{aligned}
&\underset{(\sigma_1,\dots,\sigma_n)\in\Sigma(\mathcal{X}_1)\times\dots\times\Sigma(\mathcal{X}_n)}{\operatorname{argmin}} && J(\sigma_1,\dots,\sigma_n) && \\
&\text{subj. to} && \sigma_i(0) = \mathbf{x}_i^{\text{init}} \text{ and } \sigma_i(1) \in X_i^{\text{goal}} && \forall i = 1,\dots,n \\
& && \sigma_i(\alpha) \in \mathcal{X}_i^{\text{free}} && \forall i = 1,\dots,n,\ \forall \alpha \in [0,1] \\
& && D_i(\sigma_i(\alpha), \sigma_i'(\alpha), \sigma_i''(\alpha), \dots) && \forall i = 1,\dots,n,\ \forall \alpha \in [0,1] \\
& && R_i(\sigma_i(\alpha)) \cap R_j(\sigma_j(\alpha)) = \emptyset && \forall i,j = 1,\dots,n;\ i \neq j,\ \forall \alpha \in [0,1]
\end{aligned}$$

Compared to the single-robot path planning formulation, here a) the optimization is over multi-robot paths and b) a new set of constraints that prohibit the bodies of any two robots to intersect while executing the paths is added.

Analogically, to the single-robot case, the problem of motion planning can be formulated in trajectory planning framework, where the notion of time is explicitly considered. This allows to model obstacles that change in time and impose constraints on the motions that involve differentials with respect to time. The multi-robot trajectory planning problem is to find n trajectories $\pi_i : [0,T] \to \mathcal{X}_i$, one for each robot $i = 1,\dots,n$ with T being the planning horizon. The obstacle-free workspace can change in time and thus it is modeled using a function $\mathcal{W} : [0,T] \to \mathcal{P}(\mathbb{R}^d)$. The obstacle-free region of configuration space of robot i is then defined as $\mathcal{X}_i^{\text{free}}(t) := \{\mathbf{x} : \mathbf{x} \in \mathcal{X}_i \text{ and } R_i(\mathbf{x}) \subset \mathcal{W}(t)\}$. Let $\Pi(\mathcal{X}, T)$ denote the set of all functions $[0,T] \to \mathcal{X}$. Further, let $J : \Pi(\mathcal{X}_1) \times \dots \times \Pi(\mathcal{X}_n) \to \mathbb{R}$ be a cost functional. The multi-robot trajectory planning problem can be then formally defined as follows:

Problem 4 (multi-robot trajectory planning)**.** Given n quadruples $(\mathcal{X}_i^{\text{free}}, \mathbf{x}_i^{\text{init}}, X_i^{\text{goal}}, D_i)_{i=1,\dots n}$ and a cost functional J, find $\overline{\pi}^* =$

$$\begin{aligned}
&\underset{(\pi_1,\dots,\pi_n)\in\Pi(\mathcal{X}_1,T)\times\dots\times\Pi(\mathcal{X}_n,T)}{\operatorname{argmin}} && J(\pi_1,\dots,\pi_n) && \\
&\text{subj. to} && \pi_i(0) = \mathbf{x}_i^{\text{init}} \text{ and } \pi_i(T) \in X_i^{\text{goal}} && \forall i = 1,\dots,n \\
& && \pi_i(t) \in \mathcal{X}_i^{\text{free}}(t) && \forall i = 1,\dots,n,\ \forall t \in [0,T] \\
& && D_i(\pi_i(t), \pi_i'(\alpha), \pi_i''(t), \dots) && \forall i = 1,\dots,n,\ \forall t \in [0,T] \\
& && R_i(\pi_i(t)) \cap R_j(\pi_j(t)) = \emptyset && \forall i,j = 1,\dots,n;\ i \neq j,\ \forall t \in [0,T]
\end{aligned}$$

Notice that multi-robot path planning can be seen as a special case of multi-robot trajectory planning where $T = 1$ and $\mathcal{X}_i^{\text{free}}(t)$ is constant in t for each robot i.

2.2.2.1 Complexity

Multi-robot path planning and trajectory planning are known to be computationally intractable [Hopcroft et al., 1984, Solovey and Halperin, 2015, Spirakis and Yap, 1984]. In particular, finding coordinated collision-free paths for multiple circular robots moving amidst static polygonal obstacles is known to be strongly NP-hard [Spirakis and Yap, 1984]; the same task involving rectangular robots in an empty rectangular room is known to be in PSPACE-hard [Hopcroft et al., 1984]. These works make use of different sizes of the robots to show the hardness of their coordination. More recently, Solovey and Halperin [2015] shown that the problem is PSPACE-hard also for the case of identically-shaped unit-square robots moving among polygonal obstacles.

2.2.2.2 Solution Techniques

Existing techniques for finding solutions to multi-robot path planning and trajectory planning problems can be categorized along two axes.

First, they can be divided into two groups based on the structure of the state space that is searched: a) *coupled approaches* see the multi-robot system as one composite robot with many degrees of freedom and compute a solution by planning a path in a joint configuration space of all robots, while b) *decoupled approaches* plan for each robot independently and then apply some strategy to resolve conflicts between the paths of individual robots.

Second, they can be divided into two groups based on how is the computation distributed and what information must be shared by the individual robots in order to find a coordinated plan for them: a) *centralized approaches* rely on a central computer that gathers information such as the goal, cost functional and differential constraints on trajectory of each robot and then using this information compute a joint plan for all the robots, b) *decentralized approaches*, on the other hand, do not require a central component and compute the coordinated solution by message passing between the individual robots such that the goal, the cost function and constraints on feasibility stay private to each robot.

Coupled planning approaches

The coupled approach is based on a translation of a given multi-robot path planning problem into a (single-robot) path planning in a joint configuration space $\overline{\mathcal{X}}_{\text{free}}$ constructed as a Cartesian product of configuration spaces of individual robots $\overline{\mathcal{X}}_{\text{free}} \subset \mathcal{X}_1^{\text{free}} \times \ldots \times \mathcal{X}_n^{\text{free}}$.

When $\mathbf{x} \in \overline{\mathcal{X}}_{\text{free}}$ is a configuration from a joint configuration space, then $[\mathbf{x}]_i$ will denote a function $\overline{\mathcal{X}}_{\text{free}} \to \mathcal{X}_i^{\text{free}}$ that maps a joint configuration $\mathbf{x} \in \mathcal{X}_{\text{free}}$ into a partition of the joint configuration $\mathbf{x}_i \in \mathcal{X}_i^{\text{free}}$ representing the configuration of robot i. Analogically, if $\overline{\sigma} : [0,1] \to \overline{\mathcal{X}}_{\text{free}}$ is a path in joint configuration space, then function $[\overline{\sigma}]_i : ([0,1] \to \overline{\mathcal{X}}_{\text{free}}) \to ([0,1] \to \mathcal{X}_i^{\text{free}})$ defined as $[\sigma']_i(\alpha) := [\sigma'(\alpha)]_i$ maps a path σ in the joint configuration space to a partition of the path in the configuration space of robot i.

A multi-robot path planning problem $P = ((\mathcal{X}_i^{\text{free}}, \mathbf{x}_i^{\text{init}}, X_i^{\text{goal}}, D_i)_{i=1,\ldots n}, J)$ can be translated into a path planning problem $\overline{P} = (\overline{\mathcal{X}}_{\text{free}}, \overline{\mathbf{x}}_{\text{init}}, \overline{X}_{\text{goal}}, \overline{D}, \overline{J})$ as follows. The free configuration space is so-called joint configuration space constructed as Cartesian product of configuration spaces of individual robots and the configurations where any two robots are in collision are removed:

$$\overline{\mathcal{X}}_{\text{free}} := \{\mathbf{x} : \mathbf{x} \in \mathcal{X}_1^{\text{free}} \times \ldots \times \mathcal{X}_n^{\text{free}} \text{ and } \forall_{ij,\, i \neq j}\ R_i([\mathbf{x}]_i) \cap R_j([\mathbf{x}]_j) = \emptyset\}.$$

The initial joint configuration is simply an n-tuple of initial configuration of each robot

$$\overline{\mathbf{x}}_{\text{init}} := (\mathbf{x}_1^{\text{init}}, \ldots, \mathbf{x}_n^{\text{init}})$$

and the goal region is a Cartesian product of goal regions of individual robots

$$\overline{X}_{\text{goal}} := X_1^{\text{goal}} \times \ldots \times X_n^{\text{goal}}.$$

The differential constraints on the joint path are satisfied if the constraints are satisfied on path of each robot

$$\overline{D}(\overline{\sigma}) := \forall_{i=1,\ldots,n}\, D_i([\overline{\sigma}]'_i, [\overline{\sigma}]''_i, \ldots)$$

and the cost functional is simply defined as follows

$$\overline{J}(\overline{\sigma}) := J([\overline{\sigma}]_1, \ldots, [\overline{\sigma}]_n).$$

If $\overline{\sigma}^*$ is a solution of problem to the single-robot path planning problem $\overline{P}$, then $[\overline{\sigma}^*]_1, \ldots, [\overline{\sigma}^*]_n$ are n paths constituting a solution to the multi-robot path planning problem P. The same approach can be used to translate a multi-robot trajectory planning problem to a single-robot trajectory planning problem.

A solution to the path planning problem $\overline{P}$ can be found using any of the general methods for single-robot path planning described in Section 2.1. Most existing coupled methods for multi-robot path planning [Standley, 2010, Standley and Korf, 2011, Wagner and Choset, 2015] take the search-based approach to path planning in joint configuration space, i.e., they discretize the joint configuration space and subsequently search for the minimum cost path in such a discretization. The joint configuration space is discretized in form of a graph, whose vertices represent joint configurations and the edges represent joint moves that move robots from one joint configuration to another. The main advantage of the coupled methods is their ability to find a joint path that is optimal with respect to the given discretization. However, their main disadvantage is the inherent poor scalability: for a fixed graph discretization of the configuration space of a single robot, the number of vertices in the discretization of the joint configuration space grows exponentially with the number of robots. Since the worst-case time complexity of minimum-cost path search in a graph using Dijkstra's algorithm is quadratic in the number of vertices, naive search in joint configuration space is in practice unable to solve instances with more than a few robots.

The practical performance of the search in the joint configuration space can be improved by employing A* search algorithm [Hart et al., 1968] equipped with a well-informed admissible heuristics to focus the exploration of the state space. Often, the cost function is additively separable, i.e., it is defined as a sum of costs over the individual paths $J(\sigma_1, \ldots, \sigma_n) := J_1(\sigma_1) + \ldots + J_n(\sigma_n)$. For such a cost function, one can use a heuristic function defined as $h(\mathbf{x}) := \mathrm{opt}([\mathbf{x}]_1, X_1^{\mathrm{goal}}, \mathcal{X}_1^{\mathrm{free}}, J_1) + \ldots + \mathrm{opt}([\mathbf{x}]_n, X_n^{\mathrm{goal}}, \mathcal{X}_n^{\mathrm{free}}, J_n)$, where $\mathrm{opt}(\mathbf{x}, X_{\mathrm{goal}}, \mathcal{X}_{\mathrm{free}}, J)$ denotes the cost of an optimal path from state $\mathbf{x}$ to the goal region X_{goal} in the configuration space $\mathcal{X}_{\mathrm{free}}$ under the cost function J. Such a heuristic function can be in practice implemented by precomputing an optimal single-robot policy for each robot i from all vertices in the discretized configuration space of robot i to the robot's goal region using the given cost function. If the goal region is a single vertex, such a policy can be computed for instance by running Dijkstra's shortest path algorithm from the goal vertex to all other vertices. Equipping the search algorithm with an optimal policy for each robot can drastically improve its performance in situations when the robots are able to reach their targets without coordination because the heuristic function guides the search to initially explore the joint path consisting of robots' individually optimal paths. However, if all individually-optimal paths of some of the robots are in collision, then the search will have to start exploring significant portions of the joint configuration space of all robots in order to find a set of optimal collision avoiding paths, leading again to scalability issues.

Another technique that has been shown to be effective in the context of optimal multi-robot path planning is so-called operator decomposition [Standley, 2010]. If the joint configuration space is discretized naively, then each vertex of the joint configuration space discretization has an exponential number of neighboring vertices that represent all possible joint configuration that can be reached by applying each possible combination of atomic actions/motions of each robot. In result, a single expansion during A* search is computationally expensive since the heuristic estimates for a large number of neighboring vertices need to be evaluated. Operator decomposition uses a more involved discretization strategy that models the possible combinations of a robot's atomic actions using a tree of single-robot atomic moves, where at each vertex only a single robot assigns a move to a joint move. This representation allows for a

more focused search in the joint configuration space, because the heuristic estimate can be evaluated at each state of such a tree and thus some combinations of robot's moves do not ever have to be explored.

Yet another technique that allows for a significant improvement of the scalability of search in the joint configuration space is so-called independence detection [Standley, 2010, Wagner and Choset, 2015]. This technique is based on the observation that a multi-robot path planning problem P can be split in to a number of multi-robot path planning subproblems $P'_1, \ldots, P'_k$ involving fewer robots (an thus having lower dimension of the joint configuration space) given that the optimal solutions for the subproblems $P'_1, \ldots, P'_k$ do not interact with each other, i.e., the trajectories of robots in different subproblems are collision-free. For many multi-robot path path planning problems, independent subproblems can be indeed identified and solved separately. Doing so brings a benefit of reducing the maximum dimensionality of the joint configuration space that has to be searched, while maintaining the optimality of the found solution. For example, if a multi-robot path planning problem with 8 robots each having 2 degrees of freedom can be split into 4 independent subproblems, each with 2 robots, then instead of having to search a $8 \times 2 = 16$-dimensional configuration space, the solution can be found by performing four searches in $2 \times 2 = 4$-dimensional configuration space. For sparse multi-robot path planning problems such independent subproblems can be typically identified and an optimal solution can be computed efficiently. For dense multi-robot path planning on the other hand, the problem cannot be decomposed into lower-dimensional subproblems and a solution has to be sought in the original high-dimensional joint configuration space.

The OD+ID algorithm [Standley, 2010] combines operator decomposition and independence detection. It implements the idea of independence detection by explicitly forming groups with dependent robots. It starts with unary groups that contain only a single robot and computes optimal path for each robot. Then, the algorithm checks if the paths are collision-free and joins the groups whose paths are in collision into a single group and searches for a new set of optimal joint paths in each such group. The process is recursively repeated until all groups have mutually collision-free trajectories. The anytime version of OD+ID introduced by Standley and Korf [2011] first resolves conflicts suboptimally by planning an alternative path for each robot in the conflict and then attempts to improve the quality of the solution by searching in the joint configuration space of the robots in conflict.

Possible independence between robots' optimal trajectories are also exploited by M* algorithm [Wagner and Choset, 2011, 2015] that employs so-called subdimensional expansion to plan in the joint configuration space. It starts by planning in low-dimensional subspace representing configuration spaces of individual robots and increases the dimensionality when the need for coordination is detected. A version of M* algorithm called ODrM* that employs the operator decomposition strategy has also been proposed [Ferner et al., 2013].

One of the disadvantages of the coupled approach is that it is difficult to implement in a decentralized way. Although there have been recently proposed distributed, privacy-preserving state space search algorithms such as MAD-A* [Nissim and Brafman, 2014] that could be applied for distributed search in the joint configuration space, such distributed search algorithms tend to be highly communication intensive. In extreme cases, these algorithms may require to communicate one message for each expanded state, which is impractical for most distributed robotic systems where communication bandwidth is a scarce resource.

Decoupled planning approaches

Decoupled approaches do not search in the joint configuration space, but instead perform a sequence of path or trajectory planning queries in the configuration space of each robot. These techniques offer better scalability in the number of robots, but typically suffer from incompleteness, i.e., there are solvable coordination problems that these algorithms fail to solve.

A classical decoupled scheme for multi-robot path planning is prioritized planning [Erdmann and Lozano-Pérez, 1987]. In prioritized planning, each robot is assigned a unique priority and the algorithm proceeds sequentially from the highest priority robot to the lowest priority one. In each iteration, one of

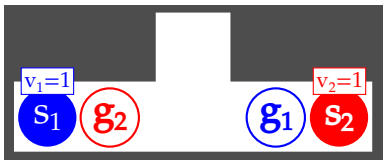
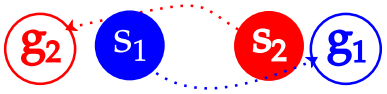
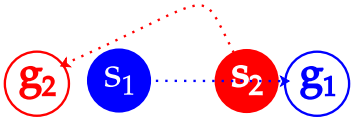


Figure 2.2.2: **Incompleteness of prioritized planning – Corridor swap scenario:** In this scenario, the robot 1 travels from s_1 to g_1 and the robot 2 travels from s_2 to g_2 in a corridor that is only slightly wider than a body of a single robot. Both robots can travel at identical maximum speeds. Observe that irrespective of which robot starts planning first, its trajectory will be in conflict with all goal-achieving trajectories of the robot that plans second.



(a) Optimal solution



(b) Solution found by prioritized planning

Figure 2.2.3: **Suboptimality of prioritized planning – Heads-on scenario:** The picture shows two robots desiring to move from s_1 to g_1 (s_2 to g_2 resp.). The top picture illustrates how an optimal solution looks like - the robots "roll" over each other and split the cost of collision avoidance equally. The bottom picture shows a solution generated by prioritized planning assuming that the robot 1 has the higher priority: robot 1 will follow a straight line path to its destination, without considering robot 2, which will have to bear the full cost of avoiding the collision.

the robots plans its trajectory such that it avoids collisions with all the higher-priority robots who follow the trajectories planned in previous iterations. Such a trajectory is found using one of the trajectory planning techniques with dynamic obstacles as described in Section 2.1.2.

Although such an approach tends to work well in uncluttered environments, it is clearly incomplete if we allow arbitrary configuration of obstacles and arbitrary start and goal locations for each robot. For an example scenario where prioritized planning fails, see Figure 2.2.2. The problem of characterization of instances that are provably solvable by prioritized planning has not been yet formally studied. A possibility to achieve completeness in multi-robot path planning on discrete roadmaps using so-called garage configurations is briefly mentioned in [LaValle, 2006], the argument is however very sketchy and no concrete algorithm with completeness guarantees is given. In Chapter 4, we provide a concrete algorithm called revised prioritized planning and a precise characterization of a class of instances in terms of allowed start and destination positions that are solvable by the algorithm. Our algorithm is shown to be complete on a class of problems where start and destination for each robot is an endpoint of a so-called well-formed infrastructure. The idea can be further extended to settings where the robots are not coordinated in batch, but rather incrementally every time when a new task is assigned to any robot in the system. In Chapter 6, we introduce an algorithm called continuous best-response approach (COBRA) that in well-formed infrastructures achieves guaranteed collision-avoidance with quadratic time-complexity in the number of robots.

Another issue that arises when decoupled methods are used is the suboptimality of generated solutions. Clearly, the quality of generated solutions is highly sensitive to the choice of priorities assigned to the robots and there is a number of works that investigate techniques for choosing good priorities for the

robots. In the context of makespan optimization, i.e. when the longest travel time is minimized, Van den Berg [2005a] proposes a heuristic that assigns a higher priority to the robots with longer travel distance. In [Bennewitz et al., 2002] randomized hill-climbing has been used to find a good priority sequence. For some problem instances, however, especially when the robots are relatively large, even the optimal priority sequence may give an unsatisfactory result – see Figure 2.2.3 for an example of such a scenario. In Chapter 7 we propose a decoupled algorithm called k-step Penalty Method (kPM) that generalizes the prioritized planning approach by employing ideas from continuous optimization that to allow to improve both quality of the returned solutions and the instance set coverage. In contrast to prioritized planning, the proposed algorithm is able to solve the two scenarios shown in Figures 2.2.2 and 2.2.3 near optimally.

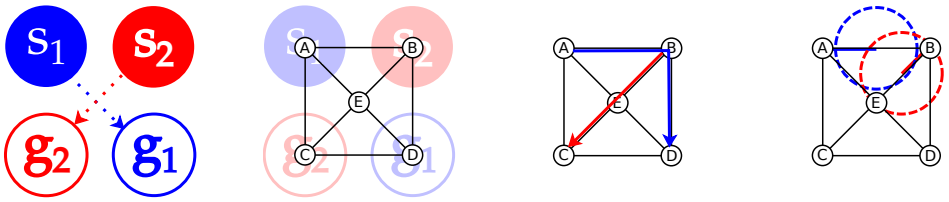
In contrast to the coupled approaches, the decoupled approaches can be implemented in a decentralized fashion in a relatively natural way. A decentralized version of prioritized planning technique proposed by Velagapudi et al. [2010] allows robots in the team to plan their paths in parallel and replan if a conflict between two paths is detected. The technique is able to utilize the distributed processing power onboard each robot to reduce the time needed to find a solution. However, the algorithm proceeds in globally-synchronized rounds, with faster-computing robots waiting at the end of each round for the longest-computing robot resulting in inefficient use of the distributed computational power. In Chapter 5, we design a generalization of the synchronized approach called asynchronous decentralized prioritized planning (ADPP) that does not rely on synchronization points, which makes it easier to implement and up to twice as fast as the synchronized version. Moreover, we show that the RPP algorithm that comes with completeness guarantees in appropriately structured environments can be also implemented using the asynchronous decentralized approach.

2.2.3 Pebble Motion Planning on Graphs

The problem of pebble motion on a graph is an abstracted formulation of the multi-robot path planning problem that has received significant treatment from the research community. In the pebble motion problem, the robots are seen as "pebbles" constrained to move on a given graph. The time is discretized and in each time step, one robot can move from one vertex to another as long as this vertex is not occupied by another robot.

The pebble motion formulation, however, has limited expressivity for modeling systems where the geometry of the robots cannot be neglected. While in the multi-robot path planning formulation, a collision is defined geometrically (the bodies of two robots overlap), in the pebble motion problem, a collision is defined on the graph (two robots occupy the same vertex). See Figure 2.2.4 for an example graph where the two notions of a collision do not coincide. For a solution of a pebble motion problem to be applicable in the context of multi-robot coordination, the geometric notion of a collision and the graph notion of a collision must coincide, which requires that the planning graph must be designed to implicitly enforce the geometric collision constraints. In particular, if we consider a system with identically shaped circular robots with radius r, then a) all vertices of the graph must be at least $2r$ apart, b) any two points on any two edges that do not share a vertex must be at least $2r$ apart, and c) the angle between edges that share a vertex must be greater than a particular value so that two robots traveling in opposite directions do not collide. Although graphs satisfying such conditions can be constructed [Vašek, 2015], they are bound to be coarse approximations of the motions that the robots can execute since the vertices need to be some minimum distance apart and the edges connecting the vertices cannot intersect. In result, the motions obtained by executing the solution of a pebble motion formulation are longer [Vašek, 2015] than the motions computed by the multi-robot motion planning approaches that explicitly work with geometric collision constraints. Besides the inability to model geometric collision constraints, the pebble motion formulation does not allow for direct modeling of heterogeneous teams or differentially constrained robots.

The pebble motion approaches are on the other hand well suited for modeling high-level coordination, where the geometric and dynamic specifics of individual robots can be abstracted away and their



(a) scenario (b) planning graph (c) pebble motion solution (d) geometric collision occurs during execution

Figure 2.2.4: Modeling multi-robot motion coordination in pebble motion framework: An example scenario where the notion of graph collisions and geometric collision do not coincide. (a) Consider scenario with two robots traveling from s_1 to g_1 and from s_2 to g_2 respectively. (b) One possible (naively chosen) discretization of motions of individual robots. (c) A valid solution in pebble motion formulation. (d) When the solution is executed with the robots having the depicted body radius, the robots will collide.

main strength is the availability of polynomial-time solution algorithms, which we are going to discuss in detail later in this section. The problem of pebble motion on graphs is formally defined as follows:

Problem 5 (Pebble motion on graphs)**.** Let $G = (V, E)$ be a graph and n be the number of robots ("pebbles") on the graph. The n-tuple $(s_1, \ldots, s_n) \in V^n$ represents the starting positions of the robots, and n-tuple $(g_1, \ldots, g_n) \in V^n$ represents goal positions of the robots. The task is to find a sequence of moves $M_i = m_1^i, \ldots, m_k^i$ for each robot i that minimizes k subject to

1. $\forall i = 1, \ldots, n : \; m_1^i = s_i$ (sequence starts at start position)
2. $\forall i = 1, \ldots, n : \; m_k^i = g_i$ (sequence finishes at goal position)
3. $\forall i = 1, \ldots, n : \forall l = 1, \ldots, k-1 : \; (m_l^i, m_{l+1}^i) \in E$ or $m_l^i = m_{l+1}^i$
   (sequence consists of moves along edges or "wait" moves)
4. $\forall i, j = 1, \ldots, n : \; i \neq j \Rightarrow \forall l = 1, \ldots, k-1 : \; m_{l+1}^i \neq m_l^j$
   (target must be unoccupied at the start of move)
5. $\forall i, j = 1, \ldots, n : \; i \neq j \Rightarrow \forall l = 1, \ldots, k : \; m_l^i \neq m_l^j$
   (target must be unoccupied at the end of move)

Some works use a relaxed version of the problem formulation and drop the fourth constraint. This version of the problem is called *abstract multi-robot path planning on graphs* [Surynek, 2009b] and allows robots to move on the graph synchronously in parallel, i.e., a robot can perform a move to a currently occupied vertex if the robot that currently occupies the vertex concurrently moves to another vertex.

2.2.3.1 Complexity

The optimal variant of the pebble motion problem on graph is known to be intractable. In particular, the problem of finding a solution to the sliding title puzzle with minimum number of title moves, which is a special case of the pebble motion problem, has been shown to be NP-hard and the decision variant to be NP-complete [Ratner and Warmuth, 1986]. More recently, Yu [2016] shown that the problem is NP-hard for all three commonly considered cost functions: minimum makespan, minimum sum of travel times, and traveled distance.

The non-optimal version of the pebble motion problem has been resolved more than two decades ago by Kornhauser et al. [1984]. The presented algorithm is P-time, solves the entire class of pebble motion on graph problems and further guarantees that the solution will consist of at most $O(n^3)$ moves.

2.2.3.2 Solution Techniques

In this section we discuss the two main classes of solution techniques for pebble motion on graphs, the optimal algorithms and non-optimal approaches.

There are a number of algorithms for finding optimal solutions to pebble motion problem. The approach *Operator decomposition and independence detection* (OD+ID) has been presented as an algorithm for optimal pebble motion on graph, but the underlying ideas can be easily generalized to less abstracted formulation of the multi-robot path planning problems and thus it is discussed in detail in Section 2.2.2.2.

Another algorithm called *Conflict-based search* (CBS) [Sharon et al., 2015] finds optimal solutions by constructing a constraint tree in which each node consists of a set of constraints, where each constraint forbids one of the agents to occupy a particular vertex of the graph at a particular time point. The search process starts with an empty set of constraints and a corresponding shortest path for each robot. For each conflict between the paths of two robots, two child nodes are added – in the first child node, one robot is constrained to avoid the vertex of conflict, while in the second node, the other robot is constrained to avoid the vertex of conflict. The cost of each node is the sum of the shortest path costs of each robot. The node is considered as a goal node if the path of each robot satisfies the constraints in the node and the paths of all robots are individually conflict-free. The experimental comparison in [Sharon et al., 2015] shows that OD+ID and CBS are incomparable in terms of scalability: in some environments OD+ID offers better performance, while in the others CBS scales better. However, one of the disadvantages of CBS in comparison with OD+ID is that the solution strategy relies on the assumption that a single robot occupies only a single vertex of the graph and thus the idea cannot be trivially applied to more general formulations of the multi-robot problem path planning problem, where robots may occupy several vertices of the underlying graph.

Recently, Yu and LaValle [2015] proposed a family of algorithms that are based on a translation of the pebble motion problem to the network flow problem, which is subsequently formulated as an integer linear program (ILP) and solved. In [Yu and LaValle, 2015] the technique is shown to be effective for different cost functions.

Although the existence of a non-optimal polynomial time algorithm for pebble motion on graphs has been known since 1984, the result fell into oblivion and has been to a large extent ignored by the research community until it was brought up to the light again by Röger and Helmert. [2012]. In the meantime, the research community developed a range of algorithms for certain subclasses of the pebble motion problem. For instance, the algorithm called *tree-based agent swapping strategy* (TASS) solves pebble motion on tree graphs and BIBOX [Surynek, 2009a] solves pebble motion on bi-connected graphs. Push&Swap [Luna and Bekris, 2011] resembles Kornhauser's approach [Kornhauser et al., 1984] but has been independently reinvented in 2011 and claimed to completely cover instances with at least two unoccupied vertices. De Wilde et al. [2014] later discovered a mistake in Push&Swap algorithm and proposed a corrected version called Push and Rotate, which is finally put in context with Kornhauser's results.

In this chapter, we gave general formulations of single-robot and multi-robot motion planning problems, reviewed related complexity results, and discussed existing solution approaches to those two problems. In the next chapter, we will introduce a specific formulation of the multi-robot motion coordination problem that will be used in the remainder of the thesis.

Chapter 3

Notation and Problem Definition

In this chapter, we will introduce some common notation and formally define several specific flavors of the multi-robot trajectory coordination problems that we will work with in the remainder of this thesis. To simplify the exposition of the proposed algorithms, we will assume a multi-robot system consisting of a number of circular omnidirectional (i.e., holonomic) robots operating in a fully-observable 2-d environment.

We consider n circular holonomic robots operating in a 2-d workspace described by a set of obstacle-free coordinates $\mathcal{W} \subseteq \mathbb{R}^2$. The maximum speed of robot i is denoted as v_i, the radius of its body is denoted by r_i. Let $R_i(\mathbf{x})$ denote the subset of $\mathcal{W}$ occupied by the body of a robot i when its center is on position $\mathbf{x}$. Note that since the robots are circular we have $R_i(\mathbf{x}) = D(\mathbf{x}, r_i)$, where $D(\mathbf{x}, r)$ denotes a closed disk centered at $\mathbf{x}$ with radius r. Each robot is assumed to be assigned a *task* that involves moving from its start position $\mathbf{s}_i$ to some goal position $\mathbf{g}_i$ and stay there. We assume that the start and goal positions of all robots are mutually disjoint, i.e., the bodies of robots do not overlap when the robots are on their start positions and when they are on their goal positions.

A path $p : [0, 1] \to \mathcal{W}$ is called *satisfying* for robot i if it starts at the robot's start position $\mathbf{s}_i$, ends at robot's goal position $\mathbf{g}_i$, and the body of the robot i whose center follows the path p always lies entirely in $\mathcal{W}$. Given a region $X \subset \mathcal{W}$, a path p is called X-avoiding if and only if $\forall \alpha \in [0, 1] : \ p(\alpha) \notin X$.

A trajectory $\pi : \ [0, \infty) \to \mathcal{W}$ is a mapping from time points to positions in the workspace and unlike a path, it carries information about how it should be executed in time. Analogically, a trajectory of the robot i is called *satisfying* if it starts at the robot's start position $\mathbf{s}_i$, eventually reaches and stays at the goal position $\mathbf{g}_i$, the body of robot i whose center follows the trajectory π always lies entirely in $\mathcal{W}$, and the robot never moves faster than its maximum speed v_i. A trajectory π is said to be X-avoiding with respect to some region $X \subset \mathcal{W}$ if and only if $\forall t \in [0, \infty) : \ \pi(t) \notin X$. A trajectory π is called $\mathbf{g}$-*terminal* if and only if $\exists t_g \, \forall t \in [t_g, \infty) : \ \pi(t) = \mathbf{g}$. The trajectories π_i, π_j of two robots i, j are said to be *conflict-free* if and only if $\forall t : \ R_i(\pi_i(t)) \cap R_j(\pi_j(t)) = \emptyset$, i.e., the bodies of the robots i, j never intersect when they follow the trajectories π_i and π_j.

We will also introduce some shorthand notations that will be used to talk about regions occupied by a different subsets of robots at their start and goal positions:

$$\begin{array}{rclcrcl} S^i & := & R_i(\mathbf{s}_i) & & G^i & := & R_i(\mathbf{g}_i) \\ S^{>i} & := & \bigcup_{j=i+1,\ldots,n} R_j(\mathbf{s}_j) & & G^{<i} & := & \bigcup_{j=1,\ldots,i-1} R_j(\mathbf{g}_j) \\ S & := & \bigcup_{j=1,\ldots,n} R_j(\mathbf{s}_j) & & G & := & \bigcup_{j=1,\ldots,n} R_j(\mathbf{g}_j). \end{array}$$

Further, we will make use of the notion of a *space-time region*: When a spatial object, such as the body of a robot, follows a given trajectory, then it can be thought of as occupying a certain region in space-time $\mathcal{T} := \mathcal{W} \times [0, \infty)$. A dynamic obstacle Δ is then a region in such a space-time $\mathcal{T}$. If $(x, y, t) \in \Delta$, then we know that the spatial position (x, y) is occupied by dynamic obstacle Δ at time t.

The function

$$R_i^{\Delta}(\pi) := \{(x, y, t) : \ t \in [0, \infty) \wedge (x, y) \in R_i(\pi(t))\},$$

maps trajectories of a robot i to regions of space-time that the robot i occupies when its center point follows given trajectory π. As a special case, let $R_i^{\Delta}(\emptyset) := \emptyset$.

After introducing the basic model and common notation, we will define a few specific flavors of the trajectory coordination problem that we will use in the remainder of this thesis.

3.1 Batch Coordination of Circular Robots

Suppose that a multi-robot system is controlled by a central component that computes the trajectories for all robots. Should a request for the coordination of trajectories of individual robots appear, the central component gathers the current position and the desired goal position of each robot and runs a multi-robot motion planner to compute a trajectory for each robot. When the computation is finished, the robots are assigned the planned trajectories and execute them. In such a scenario, the trajectory coordination occurs in batch for all robots and can be defined as follows:

Problem 6 (Batch Coordination of Circular Robots)**.** Given a workspace $\mathcal{W}$ and tasks $\langle \mathbf{s}_1, \mathbf{g}_1 \rangle, \ldots, \langle \mathbf{s}_n, \mathbf{g}_n \rangle$ for robots $1, \ldots, n$ such that $\forall i = 1, \ldots, n : \ (\mathbf{s}_i, \mathbf{g}_i) \in \mathcal{W} \times \mathcal{W}$, find trajectories $\pi_1, \ldots, \pi_n$ such that each trajectory π_i is satisfying for robot i and trajectories π_i, π_j of every two different robots i, j are mutually conflict-free.

3.2 Batch Coordination of Circular Robots in Infrastructures

In structured environments such as warehouses, factories, railroads, road networks, etc., the vehicles move only between a limited number of points of interests. For example, trains move from one station to another station. The robots in a warehouse move from one storage shelf to another shelf or to the workplace of an expedition worker. To model such structured systems, we introduce the notion of an infrastructure. An infrastructure is a pair $(\mathcal{W}, E)$, where $\mathcal{W}$ is a workspace and a finite discrete set $E \subset \mathcal{W}$ represents the endpoints in the infrastructure that are used to model the points of interest such as storage locations in a warehouse, workplaces in a factory, parking places, rail stations etc. The robots operating in an infrastructure are assumed to only move between the endpoints of the infrastructure.

The batch coordination of circular robots in infrastructures differs for the more general formulation of the problem by requiring that both the start and destination of the task of each robot are endpoints of an infrastructure:

Problem 7 (Batch Coordination of Circular Robots in Infrastructure)**.** Given an infrastructure $(\mathcal{W}, E)$ and tasks $\langle \mathbf{s}_1, \mathbf{g}_1 \rangle, \ldots, \langle \mathbf{s}_n, \mathbf{g}_n \rangle$ for robots $1, \ldots, n$ such that $\forall i = 1, \ldots, n : \ (\mathbf{s}_i, \mathbf{g}_i) \in E \times E$, find trajectories $\pi_1, \ldots, \pi_n$ such that each trajectory π_i is satisfying for robot i and trajectories π_i, π_j of every two different robots i, j are mutually conflict-free.

3.3 Decentralized Batch Coordination of Circular Robots

Consider a multi-robot system consisting of tens or hundreds of heterogeneous autonomous robots. In such a scenario, a decentralized multi-robot motion planner may be more desirable than a centralized one. With a decentralized planner, there is no central component that computes the solution, but instead, each robot runs its own instance of the algorithm and exchanges messages with the other robots according to a prescribed communication protocol. The robots use their onboard computational resources and inter-robot communication to eventually arrive to a state in which all robots hold mutually conflict-free trajectories.

An advantage of the decentralized approach in the context of multi-robot systems with heterogeneous robots is that the task specification and the kinematic, dynamic and other potentially implicit constraints on the trajectory of a particular robot remain local to that robot and do not need to be formalized or communicated, which simplifies the design of the communication protocol and allows each robot to use a custom robot-specific planner for planning its trajectory. Another advantage of such an approach is that some steps within the decentralized algorithm can be computed by different robots in parallel and thus a conflict-free solution may be computed faster.

Compared to the centralized variant, the definition of decentralized batch coordination includes the requirement that the trajectory of each robots is planned locally by the robot:

Problem 8 (Decentralized Batch Coordination of Disks)**.** Given a workspace $\mathcal{W}$ and tasks $\langle \mathbf{s}_1, \mathbf{g}_1 \rangle, \ldots, \langle \mathbf{s}_n, \mathbf{g}_n \rangle$ for robots $1, \ldots, n$ such that $\forall i = 1, \ldots, n : (\mathbf{s}_i, \mathbf{g}_i) \in \mathcal{W} \times \mathcal{W}$, find trajectories $\pi_1, \ldots, \pi_n$ such that each robot i locally plans its trajectory π_i, each trajectory π_i is satisfying for robot i, and trajectories π_i, π_j of every two different robots i, j are mutually conflict-free.

Decentralized trajectory coordination of disks in infrastructures is defined analogically, only with an added requirement on the tasks of the robots to be between the endpoints of the given infrastructure.

Problem 9 (Decentralized Batch Coordination of Disks)**.** Given an infrastructure $(\mathcal{W}, E)$ and tasks $\langle \mathbf{s}_1, \mathbf{g}_1 \rangle, \ldots, \langle \mathbf{s}_n, \mathbf{g}_n \rangle$ for robots $1, \ldots, n$ such that $\forall i = 1, \ldots, n : (\mathbf{s}_i, \mathbf{g}_i) \in E \times E$, find trajectories $\pi_1, \ldots, \pi_n$ such that each robot i locally plans its trajectory π_i, each trajectory π_i is satisfying for robot i, and trajectories π_i, π_j of every two different robots i, j are mutually conflict-free.

3.4 Online Coordination of Circular Robots in Infrastructures

Consider a multi-robot system operating in an environment modeled as an infrastructure $(\mathcal{W}, E)$. At any point during the operation of the system, any robot can be assigned a *relocation tasks* denoted $s \to g$ requesting the chosen robot to move from its current endpoint $s \in E$ to the given goal endpoint $g \in E$. We assume that a) the robot cannot be interrupted once it starts executing a particular relocation task and thus a new relocation task can be assigned to a robot only after it has reached the destination of the previously assigned task and b) the destination endpoint of a newly assigned relocation task cannot be equal to the start or destination endpoint of the task currently assigned to any of the robots. The objective is to find a trajectory for each such relocation task that, assuming that the robot will follow the trajectory precisely, will lead the robot to the specified destination endpoint without colliding with other robots operating in the system. Moreover, such trajectories should be found in a decentralized fashion without a need for a central component coordinating individual robots. The problem of online coordination is formally defined as follows:

Problem 10 (Online Coordination of Circular Robots)**.** Given a multi-robot system in an infrastructure $(\mathcal{W}, E)$ and an infinite sequence of tasks $\langle t_1, r_1, \mathbf{s}_1, \mathbf{g}_1 \rangle, \langle t_2, r_2, \mathbf{s}_2, \mathbf{g}_2 \rangle, \ldots$ assigned at time $t_1, t_2, \ldots$ to robots $r_1, r_2, \ldots$ assuming the each task $\langle t_i, r_i \mathbf{s}_i, \mathbf{g}_i \rangle$ satisfies

1. the task was assigned to a robot only when the previous task of the robot r_i has been completed (its destination endpoint has been reached) and the start endpoint $\mathbf{s}_i$ is the destination endpoint of the task previously assigned to robot r_i and
2. $\mathbf{g}_i$ is different from all start or destination endpoints of the relocation task assigned to any of the robots at time t_i,

find trajectory π_i for the task that satisfies

1. $\pi_i(t) = \mathbf{s}_i$
2. $\exists t_{\max} \forall t \geq t_{\max} : \pi_i(t) = \mathbf{g}_i$
3. $\forall j \neq i : \pi_i$ and π_j are conflict-free.

In this chapter, we have defined the basic trajectory coordination problem and its five versions: the centralized version, the centralized version in infrastructures, the decentralized version, the decentralized version in infrastructures, and the online version. In the following chapter, we will discuss the algorithms for the centralized variant of the problem.

Chapter 4

Revised Prioritized Planning

In this chapter, we analyze the prioritized planning approach, a popular decoupled technique for batch coordination of multiple mobile robots (Problem 6). We provide a novel analysis of limits of this technique and show under what circumstances is the classical prioritized planning bound to fail. Consequently, we propose a revision of this algorithm that enables a compact characterization of a large class of problem instances that are provably solvable by the algorithm. Further, in the case of batch coordination in infrastructures (Problem 7), we formulate a novel static condition on the structure of the infrastructure – we say that the infrastructure must be *well-formed* – that is sufficient to guarantee that revised prioritized planning will always succeed and compute coordinated motions between endpoints of the infrastructure in polynomial time.

4.1 Classical Prioritized Planning

A straightforward approach to solve the batch version of the circular robot coordination problem is to see all robots in the system as one composite robot with many degrees of freedom and use some path planning algorithm to find a path for such a composite robot, which represents the joint path for all individual robots. However, the size of the joint configuration space that has to be searched during the planning is exponential in the number of robots and thus this approach quickly becomes impractical if one wants to plan for more than a few robots. A pragmatic approach that is often useful even for large multi-robot teams is prioritized planning. The idea of prioritized planning has been first articulated by Erdmann and Lozano-Pérez [1987]. Since the quality of the returned solution depends on the order in which the robots plan, later works such as [van den Berg and Overmars, 2005a, Bennewitz et al., 2002] focused on heuristics for choosing a suitable priority sequence for the robots.

In classical prioritized planning, each robot is assigned a unique priority. The trajectories for individual robots are then planned sequentially from the highest priority robot to the lowest priority one. For each robot, a trajectory is planned that avoids both the static obstacles in the environment and the higher-priority robots moving along the trajectories planned in the previous iterations.

Algorithm 5 lists the pseudocode of classical prioritized planning. The algorithm iterates over the robots, starting from the highest-priority robot 1 to the lowest-priority robot n. During i-th iteration the algorithm computes a trajectory for robot i that avoids the space-time regions occupied by robots $1, \ldots, i-1$. The trajectory of the robot i is computed in `Best-traj(`$\mathcal{W}', \Delta$`)` function, which returns a trajectory for robot i such that the body of the robot stays inside the static workspace $\mathcal{W}'$ and avoids dynamic regions Δ occupied by other robots. Such a function would be in practice implemented using some application-specific technique for motion planning with dynamic obstacles, e.g., time-extended roadmap planner described in Section 4.4, [van den Berg and Overmars, 2007], or [Narayanan et al., 2012].

Algorithm 5: Classical Prioritized Planning

```
1 Algorithm PP
2     Δ ← ∅;
3     for i ← 1 . . . n do
4         π_i ←Best-traj(W, Δ);
5         if π_i = ∅ then
6             report failure and terminate
7         Δ ← Δ ∪ R_i^Δ(π_i);
8 Function Best-traj(W', Δ)
9     return optimal satisfying trajectory for robot i in W' that avoids regions Δ if it exists,
      otherwise return ∅
```

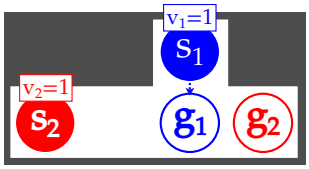


Figure 4.1.1: **Type A conflict:** Robot 1 travels from $\mathbf{s}_1$ to $\mathbf{g}_1$ and robot 2 travels from $\mathbf{s}_2$ to $\mathbf{g}_2$. The gray areas represent obstacles. Both robots can travel at identical maximum speed. The trajectory for robot 1 is planned first and the straight line trajectory from $\mathbf{s}_1$ to $\mathbf{g}_1$ at the maximum speed is found. Consequently, all satisfying trajectories of robot 2 are in Type A conflict with robot 1.

Properties

Classical prioritized planning terminates either with success or with failure in at most n iterations. The successful termination occurs in exactly n iterations if in each iteration i a valid trajectory for robot i has been found. The termination with failure occurs if there exists a robot for which no satisfying trajectory that avoids higher-priority robots can be found.

When the algorithm terminates successfully, each robot is assigned a computed trajectory that is conflict-free with respect to the trajectories of the other robots. This follows from the fact that the trajectory planned for each robot i is conflict-free with the higher-priority robots, since it was planned to avoid collision with them and also with the lower-priority robots since their trajectories were planned to avoid collisions with the trajectory of the robot i.

Prioritized planning is in general incomplete, consider the counter-example presented earlier in Figure 2.2.2.

Let us now analyze when the classical prioritized planning algorithm is bound to fail. The algorithm fails to find a trajectory for robot i if 1) no satisfying path exists for robot i, i.e., the robot cannot reach its destination even if there are no other robots in the workspace; 2) every satisfying trajectory of the robot i is in conflict with some higher-priority robot. There are two types of conflicts that can occur between a satisfying trajectory π of the robot i and a higher-priority robot:

Type A: Occurs if the trajectory π is in conflict with a higher-priority robot which has reached and is "sitting" at its destination, i.e., it is blocked by a static higher-priority robot. Figure 4.1.1 depicts a scenario in which all satisfying trajectories of one of the robots are in Type A conflict.

Type B: Occurs if the trajectory π of robot i is in conflict with a higher-priority robot which is moving towards its destination, i.e., it is "run over" by a moving higher-priority robot. Figure 4.1.2 shows a scenario in which all satisfying trajectories of one of the robots are in Type B conflict.

A question that naturally arises is whether it is possible to restrict the class of solvable instances

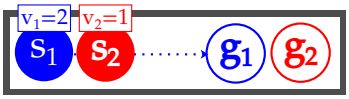


Figure 4.1.2: **Type B conflict:** Robot 1 travels from $\mathbf{s}_1$ to $\mathbf{g}_1$ and robot 2 travels from $\mathbf{s}_2$ to $\mathbf{g}_2$. The gray areas represent obstacles. Robot 1 can travel *twice as fast* as robot 2. The trajectory for robot 1 is planned first and the straight line trajectory from $\mathbf{s}_1$ to $\mathbf{g}_1$ at the maximum speed is found. The trajectory for robot 2 is planned second, but all satisfying trajectories for robot 2 are in Type B conflict with robot 1 and thus the trajectory planning for robot 2 fails.

or to alter the classical prioritized planning algorithm in such a way that there is always at least one trajectory without neither Type A nor Type B conflict for each robot.

One way to ensure that there is a satisfying trajectory without Type A conflict for every robot is to only consider instances, where each robot has a path to its goal that avoids the goal regions of all higher-priority robots. When each robot follows such a path, then they cannot be engaged in a Type A conflict, because the Type A conflict can only occur at the goal region of one of the higher-priority robots.

Unfortunately, the existence of a trajectory without Type B conflict is difficult to guarantee in classical prioritized planning algorithm, because interactions with lower-priority robots are completely ignored during trajectory planning at each iteration of the algorithm. If we want to ensure that each robot has a satisfying trajectory without Type B conflict, the trajectories of all higher-priority robots have to be planned such that the lower-priority robots are always left with some alternative trajectory that can be used to avoid the potential conflicts of this type.

One way to ensure that there is a satisfying trajectory without Type B conflict for every robot is to consider only instances in which each robot has a path to its goal that avoids start region of lower-priority robots and enforce that the trajectory of each robot will avoid the start regions of all lower-priority robots. When this is ensured, then any robot always has a fallback option to wait at its start position (since no higher-priority robot can run over its start region) until its desired path is clear of the all higher-priority robots. Thus it can always avoid Type B conflicts.

Moreover, if the robot continues by following a path that avoids the goal regions of higher-priority robots, then the resulting trajectory is also guaranteed to avoid Type A conflicts.

The result of the above analysis can be intuitively summarized as follows: If we have a trajectory coordination problem instance in which every robot can reach its goal without crossing a) the regions occupied by the lower-priority robots at their start positions and b) the regions occupied by the higher-priority robots at their goal positions, then such an instance can be in the worst-case resolved by moving the robots sequentially, one after another, to their destinations.

We can state the sufficient condition for the existence of a sequential solution formally:

Theorem 11. *Let us have a trajectory coordination problem with workspace $\mathcal{W}$ and tasks $\langle \mathbf{s}_1, \mathbf{g}_1 \rangle, \ldots, \langle \mathbf{s}_n, \mathbf{g}_n \rangle$ for robots $1, \ldots, n$. If for every robot i there exists an $S^{>i}$-avoiding and $G^{<i}$-avoiding satisfying path, then a sequential conflict-free solution can be constructed.*

Proof. We will construct the solution inductively as follows:

Induction assumption: The trajectories of robots $1, \ldots, i-1$ are satisfying and $S^{>i-1}$-avoiding.

Base step (robot 1): Robot 1 is the highest-priority robot. There are no higher-priority robots that the robot 1 needs to avoid. From our assumption there exists a path p that is satisfying for robot 1 and $S^{>1}$-avoiding. A satisfying and $S^{>1}$-avoiding trajectory for robot 1 can be simply constructed by following the path p at an arbitrary positive speed. Such a trajectory can always be constructed, therefore the algorithm will not report failure when planning for robot 1.

Induction step (robot i): From our assumption there exists a path p that is satisfying for robot i, $S^{>i}$-avoiding and $G^{<i}$-avoiding. Since all the trajectories for robots $1, \ldots, i-1$ are satisfying (i.e., eventually reach the goal and stay there), there must exist a time point $\bar{t}$ after which all robots $1, \ldots, i-1$

have reached and will stay at their goal. A satisfying and $S^{>i}$-avoiding trajectory for robot i that is conflict-free with all robots $1, \dots i-1$ can be constructed as follows:

- In interval $[0, \overline{t}]$ stay at $\mathbf{s}_i$. The trajectory cannot be in conflict with the higher-priority robots during this interval because all trajectories of robots $1, \dots, i-1$ are $S^{>i-1}$ and thus also S^i-avoiding.
- In interval $[\overline{t}, \infty]$ follow path p until the goal position $\mathbf{g}_i$ is reached. The path p avoids regions $G^{<i}$ and thus the trajectory cannot be in collision with any of the higher-priority robots $1, \dots, i-1$ because they are at their goal positions during this time interval, which the path p avoids.

Such a trajectory can always be constructed.

The trajectories of robots $1, \dots, i-1$ are satisfying and $S^{>i-1}$-avoiding, which implies that they are also $S^{>i}$-avoiding. The newly computed trajectory for robot i is satisfying and $S^{>i}$-avoiding. By taking the union of the old set of trajectories and the new trajectory, we have a set of trajectories for robots $1, \dots, i$ that are satisfying and $S^{>i}$-avoiding. □

As we can see from the constructive proof of Theorem 11, the instances that admit $S^{>i}$-avoiding and $G^{<i}$-avoiding paths for each robot can be solved by a simple sequential algorithm that navigates all robots one after another along their shortest $S^{>i}$-avoiding and $G^{<i}$-avoiding paths. Albeit simple, such an approach never lets two robots move concurrently and thus it typically generates solutions of very poor quality. The solution quality can be improved if we adopt the prioritized planning approach and find for each robot a minimum-cost $S^{>i}$-avoiding trajectory that avoids conflicts with higher-priority robots.

4.2 Revised Prioritized Planning

Using insights from the preceding discussion, we propose a Revised version of Prioritized Planning (RPP) in which a trajectory for each robot is sought so that both a) start position of all lower-priority robots are avoided and b) conflicts with higher-priority robots are avoided. The pseudocode of RPP is listed in Algorithm 6.

Algorithm 6: Revised Prioritized Planning

```
1 Algorithm RPP
2     Δ ← ∅;
3     for i ← 1 ... n do
4         S ← ⋃_{j>i} S^j
5         π_i ← Best-traj(W \ S, Δ);
6         if π_i = ∅ then
7             report failure and terminate;
8         Δ ← Δ ∪ R_i^Δ(π_i);
```

Properties

The RPP algorithm inherits the termination and soundness properties from the PP algorithm. The algorithm terminates successfully after n iterations if a trajectory for each robot has been found. The algorithm terminates with a failure at iteration i if there is a robot i for which a satisfying trajectory in $\mathcal{W} \setminus S$ has not been found.

In general, it is not guaranteed that a trajectory that avoids both start positions of lower-priority robots and regions occupied by higher-priority robots will exist for each robot. Thus the algorithm may fail to provide a solution to a solvable problem instance. Consider the example in Figure 2.2.2 once again. However, for instances characterized by the following condition, the solution is guaranteed to exist and RPP will find it.

Corollary 12. *If there is a $S^{>i}$-avoiding, $G^{<i}$-avoiding satisfying path for every robot i and a complete algorithm is used for the single-robot trajectory planning in* `Best-traj` *function, then RPP is guaranteed to terminate with a conflict-free solution.*

Proof. Consider the inductive argument from the proof of Theorem 11. The argument states that at every iteration, there exists a $S^{>i}$-avoiding satisfying trajectory for robot i that avoids all higher-priority robots. Since the single-robot planning algorithm is assumed to be complete, it cannot fail in finding such a trajectory. □

4.3 Well-formed Infrastructures

In the sequel, we will show that it is possible to design operational environments for the multi-robot system where the assumptions from Corollary 12 always hold and consequently obtain a guaranteed and tractable collision avoidance method with RPP approach. To guarantee that $S^{>i}$-avoiding and $G^{<i}$-avoiding paths will always exist, we need to constrain allowed start and goal positions for task assigned to robots. Therefore, we will make use of more structured problem formulation of batch multi-robot coordination in infrastructures (Problem 7). Recall that an infrastructure is defined as a pair $(\mathcal{W}, E)$, where $\mathcal{W}$ is the workspace (described by a set of obstacle-free coordinates $\mathcal{W} \subseteq \mathbb{R}^2$) and E is a discrete set of distinguished points from $\mathcal{W}$ called endpoints (modeling, e.g., storage locations in a warehouse, workplaces in a factory, parking places, road stops, etc.). It is assumed that the start and destination of any task assigned to a robot is one of the endpoints of the infrastructure.

Now we can define a class of *well-formed infrastructures* that satisfy the condition of having an $S^{>i}$-avoiding and $G^{<i}$-avoiding paths for every possible task of every robot: A *well-formed* infrastructure has its endpoints distributed in such a way that any robot standing on an endpoint cannot completely prevent other robots from moving between any other two endpoints. In a well-formed infrastructure, a robot is always able to find a collision-free trajectory to any other unoccupied endpoint by waiting for other robots to reach their destination endpoint and then by following a path around the occupied endpoints, which is in a well-formed infrastructure guaranteed to exist.

In the following, we will describe the idea more formally. First, let us introduce the necessary notation. Let $D(\mathbf{x}, r)$ be a closed disk centered at x with radius r and $\mathrm{int}_r\ X$ be an r-interior of set X defined as

$$\mathrm{int}_r\ X := \{\mathbf{x} : D(\mathbf{x}, r) \subseteq X\}\,.$$

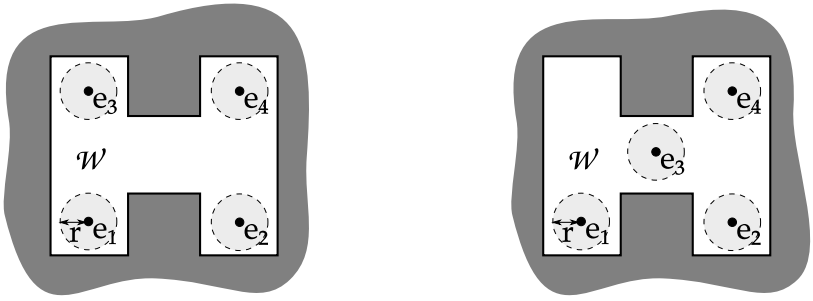
Any path that lies entirely in $\mathrm{int}_r\ X$ will have r-clearance with respect to X, i.e. any point on the path will be at minimum distance r from the closest boundary of X.

Definition 13. An infrastructure $(\mathcal{W}, E)$ is called *well-formed* for circular robots having body radii $r_1, \ldots, r_n$ if any two endpoints $\mathbf{a}, \mathbf{b} \in E$ can be connected by a path in workspace

$$\mathrm{int}_{\overline{r}} \left(\mathcal{W} \setminus \bigcup_{\mathbf{e} \in E \setminus \{\mathbf{a}, \mathbf{b}\}} D(\mathbf{e}, \overline{r}) \right),$$

where $\overline{r} = \max\{r_1, \ldots, r_n\}$.

That is, there must exist a path between any two endpoints with at least $\overline{r}$-clearance with respect to static obstacles and at least $2\overline{r}$-clearance to any other endpoint. See Figure 4.3.1 for an illustration.



(a) Well-formed infrastructure: The workspace $\mathcal{W}$ and endpoints $\{e_1, e_2, e_3, e_4\}$ for robots having radius r form a well-formed infrastructure.

(b) Ill-formed infrastructure: The workspace $\mathcal{W}$ and endpoints $\{e_1, e_2, e_3\}$ do not form a valid infrastructure because there is no path from e_1 to e_2 with $2r$-clearance to e_3 for a robot having radius r.

Figure 4.3.1: Well-formed and ill-formed infrastructure

The notion of well-formed infrastructure formalizes the structural property typically witnessed in man-made environments that are intuitively designed to allow efficient transit of multiple persons or vehicles. In such environments, the endpoint locations where people or vehicle are allowed to stop for a long time are separated from the transit area that is reserved for travel between these locations.

In a road network, for example, the endpoints would be the parking places. The system of roads is then built in such a way that any two parking places are reachable without crossing any other parking place. A similar structure can be witnessed in offices or factories. The endpoints would be all locations, where people may need to spend longer periods of time, e.g. surroundings of the work desks or machines. As we know from our everyday experience, work desks and machines are typically given enough free room around them so that a person working at a desk or a machine does not obstruct people moving between other desks or machines. We can see that real-world environments are indeed often designed as well-formed infrastructures.

Roadmap for Well-formed Infrastructure

The notion of a well-formed infrastructure can be extended to discretized representations of the robots' common workspace as follows.

Definition 14. A graph $G = (V, L)$ is a roadmap for a well-formed infrastructure $(\mathcal{W}, E)$ and robots with radii $r_1, \dots r_n$ if $E \subseteq V$ and any two different endpoints $\mathbf{a}, \mathbf{b} \in E$ can be connected by a path $p = \mathbf{l}_1, \dots, \mathbf{l}_k$ in graph G such that

$$\underset{i=1,\dots,k}{\forall} \operatorname{line}(\mathbf{l}_i) \subseteq \operatorname{int}_{\overline{r}} \left(\mathcal{W} \setminus \bigcup_{\mathbf{e} \in E \setminus \{\mathbf{a},\mathbf{b}\}} D(\mathbf{e}, \overline{r}) \right),$$

where $\overline{r} = \max\{r_1, \dots, r_n\}$ and $\operatorname{line}(\mathbf{l})$ is the set of points lying on a straight line between the vertices that the edge $\mathbf{l}$ connects.

Analogically to the well-formed infrastructure, we require that the roadmap contains a path that connects any two endpoints with at least $\overline{r}$ clearance to static obstacles and $2\overline{r}$ clearance to other endpoints.

Completeness of RPP in Well-formed Infrastructures

Suppose a particular workspace $\mathcal{W}$ and a set of endpoints E form a well-formed infrastructure for robots having radius r. An instance of multi-robot trajectory coordination problem in such an infrastructure

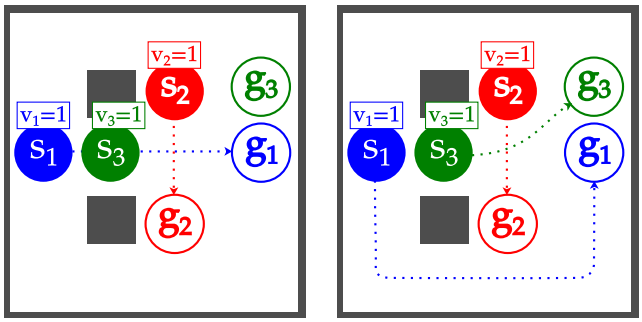


Figure 4.3.2: **Scenario, where PP fails, but RPP succeeds.** Robot 1 travels from $\mathbf{s}_1$ to $\mathbf{g}_1$, robot 2 travels from $\mathbf{s}_2$ to $\mathbf{g}_2$, robot 3 travels from $\mathbf{s}_3$ to $\mathbf{g}_3$. The gray areas represent obstacles. All robots can travel at an identical maximum speed. **Left:** In PP, the straight-line trajectory between $\mathbf{s}_1$ and $\mathbf{g}_1$ at maximum speed is found for robot 1. Similarly, the straight line trajectory at maximum speed is found also for robot 2. Consequently, all satisfying trajectories for robot 3 are in Type B conflict with robot 1 or robot 2 and thus trajectory planning for robot 3 fails. **Right:** In RPP, a trajectory that avoids the region occupied by robot 3 at its start position is found for robots 1 and 2. This allows robot 3 to avoid collisions with robot 2 by waiting at its start position and then by following the shortest path to $\mathbf{g}_3$.

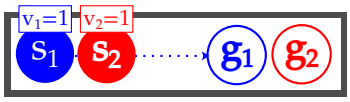


Figure 4.3.3: **Scenario, where PP succeeds, but RPP fails.** Robot 1 travels from s_1 to g_1, robot 2 travels from s_2 to g_2. The gray areas represent obstacles. Assume that both robots can travel at the same maximum speed. Robot 1 searches for a trajectory that avoids the start position of robot 2; such a trajectory does not exist and thus RPP terminates with failure. Note that PP will successfully find a solution to this instance: Robot 1 will plan the trajectory that follows the straight line at maximum speed. Robot 2 will search for a trajectory that avoids the trajectory of robot 1 and finds that it suffices to travel at maximum speed to its goal g_2 along a straight line.

would then involve a number of robots traveling from one endpoint to another so that each endpoint is used by at most one robot. From the well-formed infrastructure property, we know that for each robot there is a path p from its start endpoint to its goal endpoint that avoids all other endpoints in the infrastructure with $2r$ clearance. Since all other robots' start and goal positions lie at endpoints, path p is also avoiding start and goal position of every other robot. In other words, for every robot, there is a path that is S^{-i}-avoiding and G^{-i}-avoiding, which implies that the path is also $S^{>i}$-avoiding and $G^{<i}$-avoiding. Therefore, any trajectory coordination query in which all robots move between endpoints of a well-formed infrastructure will be successfully solved by the RPP algorithm, given that a complete algorithm is used for the single robot trajectory planning.

RPP vs. PP

We have shown that there is a class of instances that RPP completely covers, but PP does not. See Figure 4.3.2 for an example of such an instance. However, outside of this class we can find instances that PP solves, but RPP does not. For an example, consider the scenario depicted in Figure 4.3.3. None of the algorithms is therefore superior to the other in terms of instance coverage. Further, since RPP avoids start regions preemptively, even when they can be safely passed through, the solutions generated

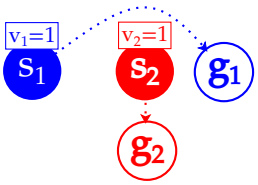


Figure 4.3.4: **Scenario, where PP finds a higher quality solution than RPP.** Robot 1 travels from $\mathbf{s}_1$ to $\mathbf{g}_1$, robot 2 travels from $\mathbf{s}_2$ to $\mathbf{g}_2$. There are no obstacles in this scenario. When RPP searches for a trajectory for robot 1 it has to avoid the start position of robot 2, resulting in the curved trajectory as depicted in the picture. On the other hand, PP generates a shorter straight-line trajectory connecting start and destination of robot 1.

by RPP tend to be slightly longer than the ones generated by PP. This is demonstrated in Figure 4.3.4. We can see that despite the theoretical guarantees of RPP, there exist situations in which PP can be a more appropriate choice than RPP.

4.4 Time-extended Roadmap Planner

For the analysis of the computational complexity of RPP and other algorithms introduced in this thesis, we will often need to assume a specific procedure for finding optimal trajectories in dynamic environments. Therefore, in the following section, we will introduce a search-based trajectory planner for dynamic environments, which we will refer to as Time-Extended Roadmap Planner (TERP), and derive its properties.

Time-extended roadmap planner is a simple trajectory planner for a holonomic point-robot constrained to move at maximum speed $v_{\max}$ in 2-d or 3-d workspace $\mathcal{W}$. The planner returns a trajectory π^* from the given start configuration $\mathbf{x}_{\text{init}}$ at time t_{init} to a given goal vertex $\mathbf{x}_{\text{goal}}$ optimizing the time to reach the goal, i.e., the cost function $\hat{c}^{\text{arr}}(\pi) := \min_{t \in [0,\infty)} t' \geq t : \pi(t) = \mathbf{x}_{\text{goal}}$. The planner uses a roadmap (i.e., a graph representation of the static 2-d or 3-d workspace) and extends it with a discretized time dimension. The vertices and edges of the roadmap are assumed to be collision-free with respect to static obstacles. The dynamic obstacles will represent other robots moving along their trajectories and are therefore assumed to be circular objects following known piecewise linear trajectories. The time-extended roadmap is computed in such a way that each edge in the time-extended roadmap is collision-free with respect to both static and dynamic obstacles. The time-extended roadmap is subsequently searched to obtain a shortest path from $(\mathbf{x}_{\text{init}}, t_{\text{init}})$ to $(\mathbf{x}_{\text{goal}}, \infty)$ using Dijkstra's shortest path algorithm or A* with a suitable admissible heuristic.

Let $G = (V, L)$ be a spatial roadmap representing an arbitrary discretization of the static workspace $\mathcal{W}$, where $V \subseteq \mathcal{W}$ represent the discretized configurations from $\mathcal{W}$ and $L \subseteq V \times V$ represent possible straight-line transitions between the discretized configurations in the roadmap. The dynamic obstacles are represented by a set of configurations $\Delta \subset \mathcal{W} \times [0, \infty)$. The time-extended roadmap constructed over a spatial roadmap G, avoiding dynamic obstacles Δ, with time discretization step δ, for robot starting at position $\mathbf{x}_{\text{init}}$ at time t_{init}, and constrained to move at maximum speed $v_{\max}$ is denoted as $\text{TER}(\text{G}, \Delta, \delta, \mathbf{x}_{\text{init}}, \text{t}_{\text{init}}, \text{v}_{\max})$. It is a graph $\hat{G} = (\hat{V}, \hat{L})$ recursively defined as follows:

1. $(\mathbf{x}_{\text{init}}, t_{\text{init}}) \notin \Delta \Rightarrow (\mathbf{x}_{\text{init}}, t_{\text{init}}) \in \hat{V}$,

2. $\forall \mathbf{v}_1, \mathbf{v}_2, t : (\mathbf{v}_1, t) \in \hat{V} \wedge (\mathbf{v}_1, \mathbf{v}_2) \in L :$
   $\text{line}((\mathbf{v}_1, t), (\mathbf{v}_2, t')) \cap \Delta = \emptyset \Rightarrow (\mathbf{v}_2, t') \in \hat{V} \wedge ((\mathbf{v}_1, t), (\mathbf{v}_2, t')) \in \hat{L},$
   where $t' = \left\lceil \frac{|\mathbf{v}_2 - \mathbf{v}_1|}{v_{\max} \cdot \delta} \right\rceil \delta,$

3. $\forall (\mathbf{v}, t) \in \hat{V} :$
   $\text{line}((\mathbf{v}, t), (\mathbf{v}, t + \delta)) \cap \Delta = \emptyset \Rightarrow (\mathbf{v}, t + \delta) \in \hat{V} \wedge ((\mathbf{v}, t), (\mathbf{v}, t + \delta)) \in \hat{L},$

where $\text{line}(\mathbf{x}, \mathbf{y})$ represents the set of points lying on the line $\mathbf{x} \rightarrow \mathbf{y}$. Each vertex $(\mathbf{x}, t) \in \hat{V}$ of the time-extended roadmap represents a possible collision-free position $\mathbf{x}$ for the robot at timepoint t. Similarly, each edge $((\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2)) \in \hat{L}$ represents a partial trajectory for the robot that starts at position $\mathbf{x}_1$ at time t_1 and follows the straight-line path from $\mathbf{x}_1$ to $\mathbf{x}_2$ at a constant speed v chosen such that $v \leq v_{\max}$ and the robot reaches configuration $\mathbf{x}_2$ at time t_2. Note that in order to make each space-time edge end at a time point that is a multiple of the time discretization step δ, some of the edges must be traveled at a slower speed than $v_{\max}$.

There are different objective criteria that can be considered. A commonly used cost function is the arrival time to the goal location defined as $\hat{c}^{\text{arr}}(\pi, \mathbf{g}) := \min_{t \in [0,\infty)} \forall t' \geq t : \pi(t) = \mathbf{g}$, i.e. the time after which the robot following the trajectory reaches the goal position $\mathbf{g}$ and will never leave it again.

The shortest path is then sought from the initial vertex $(\mathbf{x}_{\text{init}}, t_{\text{init}})$ to a vertex $(\mathbf{g}, t_g)$ that satisfies $\mathbf{g} \in \hat{X}_{\text{goal}}(t_g)$. The resulting trajectory is then constructed by concatenating the space-time edges (partial trajectories) comprising the shortest path in the time-extended roadmap $\hat{G}$.

Properties

Since the time-extended roadmap $\hat{G}$ is searched using Dijkstra's shortest path algorithm, the method returns the minimum-cost collision-free path to the goal region that can be constructed from the partial trajectories contained in the roadmap.

An important property of this planning approach is that if the dynamic obstacles stop changing at some point in time, a bound on the cost of the resulting trajectory can be obtained.

Lemma 15. *If the set of dynamic obstacles $\Delta(t)$ remains constant after time point t' and an optimal trajectory π^* from $(\mathbf{x}_{\text{init}}, t_{\text{init}})$ to $(\mathbf{x}_{\text{goal}}, \infty)$ exists in time-extended roadmap $\hat{G} = TER(G = (V, E), \Delta, \delta, \mathbf{x}_{\text{init}}, t_{init}, v_{\max})$, then its cost can be bounded by $\hat{c}^{\text{arr}}(\pi^*, \mathbf{x}_{\text{goal}}) \leq \max(t_{\text{init}}, t') + \left\lceil \frac{d}{v_{\max} \cdot \delta} \right\rceil \delta |V|$, where d is the length of longest edge in the spatial roadmap G.*

Proof. From assumption, we know that there is a trajectory π^* that is composed of edges from time-extended roadmap $\hat{G}$, satisfies $\pi^*(t_{\text{init}}) = \mathbf{x}_{\text{init}}$, $\pi^*(\infty) = \mathbf{x}_{goal}$, $\forall t \in [t_{\text{init}}, \infty) : \pi^*(t) \notin \Delta$ and is optimal with respect to $\hat{c}^{\text{arr}}(\pi^*, \mathbf{x}_{\text{goal}})$. Let t^{arr} denote the earliest time point when the trajectory reaches the goal position and stays there, i.e. $t^{\text{arr}} := \min_{t \in [0,\infty)} \forall t_1 \geq t : \pi(t_1) = x_{\text{goal}}$. Further, let $t'' := \max(t_{\text{init}}, t')$ and observe that $t'' \geq t'$. We analyze two cases:

1. Assume $\pi^*(t'') = \mathbf{x}_{\text{goal}}$. From assumption, we know that after t'' the dynamic obstacles do not change and thus if $\mathbf{x}_{\text{goal}} \notin \Delta(t'')$, then $\forall t \geq t'' : \mathbf{x}_{\text{goal}} \notin \Delta(t)$. Consequently, the optimal trajectory will stay at $\mathbf{x}_{\text{goal}}$ after t'' since choosing otherwise would increase the objective criterion. In result, $t^{\text{arr}} \leq t''$.

2. Assume $\pi^*(t'') \neq \mathbf{x}_{\text{goal}}$. Let us denote $\mathbf{x} := \pi^*(t'')$. Since π^* is an optimal trajectory, then the sub-trajectory of π^* on the interval $[t'', t^{\text{arr}}]$ denoted π_1 is a trajectory from $\mathbf{x}$ starting at t'' to $\mathbf{x}_{\text{goal}}$ at t^{arr} optimal with respect to $\hat{c}^{\text{arr}}(\pi_1, \mathbf{x}_{\text{goal}})$. Since the set of obstacles Δ does not change after t'', the optimal trajectory can be constructed by following an optimal spatial path

at maximum possible speed. This path can be found on a spatial roadmap graph $G' = (V', L')$, where $V' = \{\mathbf{v} : \mathbf{v} \in V \text{ and } \mathbf{v} \notin \Delta\}$ and $L' = \{(\mathbf{s}, \mathbf{e}) : (\mathbf{s}, \mathbf{e}) \in L \text{ and } \text{line}(\mathbf{s}, \mathbf{e}) \cap \Delta = \emptyset\}$ using cost function that assigns to each edge a cost that corresponds to the travel time over the same edge in the time-extended roadmap, i.e. $c((\mathbf{s}, \mathbf{e})) := \left\lceil \frac{|\mathbf{e}-\mathbf{s}|}{v_{\max} \cdot \delta} \right\rceil \delta$. The cost of the optimal path from $\mathbf{x}$ to $\mathbf{x}_{\text{goal}}$ in graph G' denoted as p^* is equal to the duration of trajectory π_1. Since p^* is optimal, it only visits each vertex at most once. Therefore, the path p^* will be composed of at most $|V|$ edges. Further, the cost of each edge is bounded by $\left\lceil \frac{d}{v_{\max} \cdot \delta} \right\rceil \delta$. Consequently, the cost of the path p^* is bounded by $\left\lceil \frac{d}{v_{\max} \cdot \delta} \right\rceil \delta |V|$, which in turn corresponds to the duration of trajectory π_1. Once the point $\mathbf{x}_{\text{goal}}$ is reached, the optimal trajectory will stay at $\mathbf{x}_{\text{goal}}$ because from our assumption, we have $\forall t \geq t'' : \mathbf{x}_{\text{goal}} \notin \Delta$ and doing otherwise would contradict the optimality of π^*. In result, $t^{\text{arr}} \leq t'' + \left\lceil \frac{d}{v_{\max} \cdot \delta} \right\rceil \delta |V|$.

In both cases, we have $t^{\text{arr}} \leq t'' + \left\lceil \frac{d}{v_{\max} \cdot \delta} \right\rceil \delta |V|$ and consequently we conclude $\hat{c}^{\text{arr}}(\pi^*, \mathbf{x}_{\text{goal}}) \leq \max(t_{\text{init}}, t') + \left\lceil \frac{d}{v_{\max} \cdot \delta} \right\rceil \delta |V|$. □

Complexity

The computational complexity of planning using TERP will depend on the complexity of checking whether an edge or a vertex in a time-extended roadmap is in a collision with dynamic obstacles. In this and the following chapters, we will need to plan an optimal trajectory for a disc robot that avoids other robots following piecewise linear trajectories. Therefore, we will consider dynamic obstacles to be discs that move along known piecewise-linear trajectories. For practical reasons, we will assume that the trajectories of moving robots switch from one linear segment to another only at a time points that are multiples of timestep δ. If there is a timepoint after which the moving obstacles stop changing, then the worst-case computational complexity of finding a collision-free trajectory can be derived:

Lemma 16. *If the set of dynamic obstacles $\Delta(t)$ composed of m discs moving along piecewise linear trajectories remains constant after time point t' and an optimal collision-free trajectory π^* from $(\mathbf{x}_{\text{init}}, t_{\text{init}})$ to $(\mathbf{x}_{\text{goal}}, \infty)$ exists in time-extended roadmap $\hat{G} = TER(G = (V, E), \Delta, \delta, \mathbf{x}_{\text{init}}, t_{init}, v_{\max})$, then TERP will find π^* in time $O(m\tau k |V|^2 + \tau^2 |V|^2)$, where $\tau := (\max(0, t' - t_{\text{init}}))/\delta + k |V|$, $k := \left\lceil \frac{d}{v_{\max} \cdot \delta} \right\rceil$, and d is the length of the longest edge in the spatial roadmap G.*

Proof. From the assumption we know that feasible and optimal $\pi^* : [t_{\text{init}}, \infty) \to \mathcal{W}$ exists. Let $k := \left\lceil \frac{d}{v_{\max} \cdot \delta} \right\rceil$ be a bound on the number of time steps the longest edge in graph $\hat{G}$ spans. Using Lemma 15 we can bound the cost of π^* as $\hat{c}^{\text{arr}}(\pi^*, \mathbf{x}_{\text{goal}}) \leq \max(t_{\text{init}}, t') + k\delta |V|$ and thus also the time of arrival to goal position $t^{\text{arr}} \leq \max(t_{\text{init}}, t') + k\delta |V|$ after which the optimal trajectory π^* stays at the goal position $\mathbf{x}_{\text{goal}}$. Thus, it is sufficient to determine the part of the trajectory π^* only on the interval $[t_{\text{init}}, t^{\text{arr}}]$ denoted as π'. To find the trajectory π', we search the subgraph $\hat{G}'$ of graph $\hat{G}$ covering the interval $[t_{\text{init}}, t^{\text{arr}}]$ and defined as $\hat{G}' = (\hat{V}', \hat{L}')$ such that $\hat{V}' = \{(\mathbf{x}, t), : (\mathbf{x}, t) \in \hat{V} \text{ and } t \in [t_{\text{init}}, t^{\text{arr}}]\}$ and $\hat{L}' = \{((\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2)) : ((\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2)) \in \hat{L} \text{ and } t_1, t_2 \in [t_{\text{init}}, t^{\text{arr}}]\}$. Observe that time-extended roadmap will cover $\tau = (t^{\text{arr}} - t_{\text{init}})/\delta$ timesteps. At each time step, it can contain at most $|V|$ vertices and $|L| + |V|$ edges, thus the number of vertices in $\hat{G}'$ is bounded by $\left|\hat{V}'\right| \leq \tau |V|$ and $\left|\hat{L}'\right| \leq \tau(|L| + |V|)$. Using the bound on t^{arr}, we get $\tau \leq (\max(t_{\text{init}}, t') + k\delta |V| - t_{\text{init}})/\delta$, which can be rearranged to $\tau \leq (\max(0, t' - t_{\text{init}}) + k\delta |V|)/\delta$ and further to $\tau \leq \max(0, t' - t_{\text{init}})/\delta + k |V|$. In order to obtain π', the graph $\hat{G}'$ must be first constructed and then searched.

Construction: During the construction of the space-time subgraph $\hat{G}'$, each edge $\epsilon \in \hat{L}'$ has to be checked for collisions with moving obstacles Δ composed of m space-time regions, each representing

a disc body with radius r_j moving along piecewise linear trajectory π_j that changes segments only at time points that are multiples of δ. Deciding whether edge ϵ collides with a moving obstacle can be done in time linear in the number of time steps edge ϵ spans, since for each time step τ, a sub-segment corresponding to time step τ can be extracted both from ϵ and π_j and the collision-free property $\forall t: \ |\epsilon(t) - \pi_j(t)| \leq r_j + r_i$ can be validated by solving the corresponding quadratic equation. The collision-free property of each edge can be checked in time $O(km)$. There is at most $\tau(|L|+|V|)$ edges in $\hat{G}'$ and thus the graph $\hat{G}'$ can be constructed in time $O(km\tau(|L| + |V|))$ or $O(km\tau |V|^2)$ using $|L| \leq |V|^2$.

Search: The worst-case time complexity of Dijkstra's shortest path algorithm is $O(N^2)$ [Cormen et al., 2009], where N is the number vertices of the searched graph, which is in our case $N = \tau |V|$. The time-complexity of search is therefore $O(\tau^2 |V|^2)$. By combining construction and search we get $O(m\tau k |V|^2 + \tau^2 |V|^2)$, where $\tau := \max(0, t' - t_{\text{init}})/\delta + k |V|$ and $k := \left\lceil \frac{d}{v_{\max} \cdot \delta} \right\rceil$. □

4.5 Complexity

In order to derive the computational complexity of PP and RPP algorithms, we need to consider a concrete implementation of routine `Best-traj`. We will assume that the best response trajectories are computed using the time-extended roadmap planner as described in Section 4.4 on a given spatial roadmap $G = (V, L)$.

Recall that the number of robots is denoted n. Further, the number of vertices in roadmap G is denoted as v, the length of the longest edge in the roadmap G is denoted as d, the maximum speed of the robot is $v_{\max}$, the step of time discretization in time-extended roadmap is δ, and k is the number of time steps of the longest edge in time-extended roadmap computed as $k := \left\lceil \frac{d}{v_{\max} \cdot \delta} \right\rceil$. Then, we can show that the worst-case computational complexity of both PP and RPP is cubic in the number of coordinated robots n as stated by the following theorem:

Theorem 17. *The worst-case asymptotic time complexity of PP and RPP with* `Best-traj` *routine implemented using TERP is* $O(n^3k^2v^4)$.

Proof. **Induction assumption:** If satisfying collision-avoiding trajectories $\pi_1 \ldots, \pi_i$ exist for robots $1, \ldots, i$ on the roadmap G, then at time $t \geq ik\delta v$ all robots $1, \ldots, i$ will be at their goal positions, i.e. $\forall j = 1, \ldots, i: \ \pi_j(t) = g_j$.

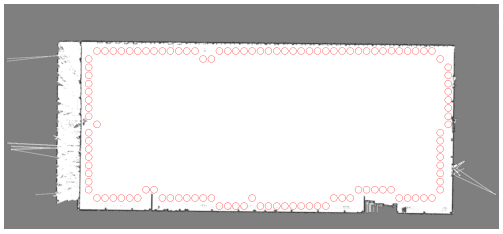
Base case (1st robot): There are no robots with higher priority than the robot 1. Therefore, the set of dynamic obstacles $\Delta = \emptyset$ and consequently it trivially stops changing at time $t' = 0$. From Lemma 15 we can get the bound on the cost of trajectory for robot 1 denoted π_1 as $\hat{c}^{\text{arr}}(\pi_1, \mathbf{g}_1) \leq \max(0, 0) + k\delta v \Rightarrow \hat{c}^{\text{arr}}(\pi_1, \mathbf{g}_1) \leq k\delta v$. From Lemma 16 and substituting $\tau = (\max(0, 0))/\delta + kv = kv$, we know that the trajectory π_1, if it exists, will be found in time $O(k^2v^4)$.

Inductive step (i-th robot): From the inductive assumption, we know that the trajectories of higher-priority robots $1, \ldots, i-1$ exist and all reach their respective goals before time $(i-1)k\delta v$. Thus, the set of dynamic obstacles Δ stops changing at $t' \leq (i-1)k\delta v$. From Lemma 15 we can get bound on the cost of trajectory π_i as

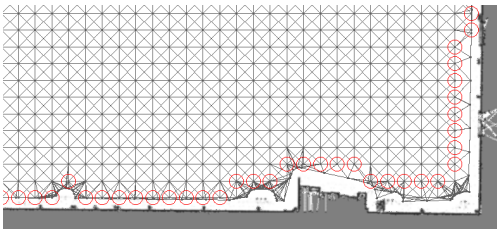
$$\hat{c}^{\text{arr}}(\pi_i, \mathbf{g}_i) \leq \max(0, t') + k\delta v$$
$$\hat{c}^{\text{arr}}(\pi_i, \mathbf{g}_i) \leq (i-1)k\delta v + k\delta v$$
$$\hat{c}^{\text{arr}}(\pi_i, \mathbf{g}_i) \leq ik\delta v,$$

and subsequently the goal arrival time t^{arr} for trajectory π_i is bounded by $ik\delta v$. From Lemma 16 and substituting $\tau = \max(0, t')/\delta + kv = t'/\delta + kv = (i-1)k\delta v/\delta + kv = ikv$, we know that the trajectory π_1, if it exists, will be found in time $O(i^2k^2v^3 + i^2k^2v^4) = O(i^2k^2v^4)$.

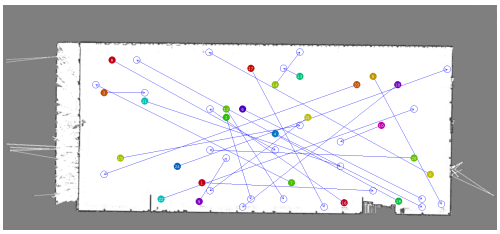
Since each robot $i = 1, \ldots, n$ computes its trajectory in time $O(i^2k^2v^4)$ and $i \leq n$, the trajectories for all n robots will be computed in time $O(n \cdot (n^2k^2v^4)) = O(n^3k^2v^4)$. □



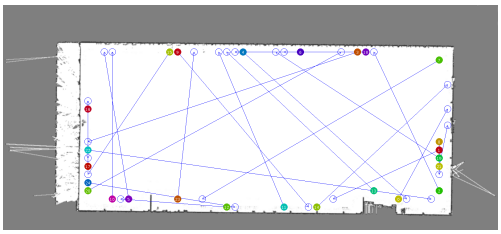
(a) *Empty Hall* Environment. The endpoints used to create a well-formed infrastructure in the environment are shown as red circles.



(b) A selected part of the roadmap used for trajectory planning.



(c) Example instance with free-formed tasks for 25 robots. Task of each robot shown in blue.



(d) Example instance with tasks for 25 robots in the well-formed infrastructure. Task of each robot shown in blue.

Figure 4.6.1: Empty Hall environment

4.6 Experimental Evaluation

In this section, we report on the results of empirical comparison between classical prioritized planning and revised prioritized planning algorithms. The *effectiveness* of the algorithms is compared by a) measuring their ability to successfully solve a set of randomly generated problem instances (i.e., instanceset coverage) in three real-world environments and b) measuring the quality of the returned solutions. Their *efficiency* is compared by measuring the runtime each algorithm requires to compute a valid solution.

These performance criteria are compared both a) on general "free-formed" instances that do not guarantee existence of $S^{>i}$-avoiding, $G^{<i}$-avoiding satisfying path for each robot and b) on instances, where robots move between endpoints of a well-formed infrastructure, i.e., on instances where the existence of an $S^{>i}$-avoiding, $G^{<i}$-avoiding satisfying path for each robot is guaranteed.

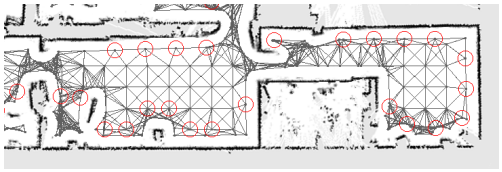
4.6.1 Environments

For experimental comparison, we use three real-world environments shown in Figure 4.6.1a, Figure 4.6.2a, and Figure 4.6.3a. For each of the environments, we generated two sets of problem instances: a) In the *free-formed tasks* instanceset, each robot is assigned a task to move from a randomly selected start position to a randomly selected goal position. b) In the *tasks in well-formed infrastructure* instanceset, we generated a set of endpoints that together with a particular roadmap discretization of the environment form a well-formed infrastructure; each robot is then assigned a random endpoint as a start position and a randomly chosen endpoint as a destination position.

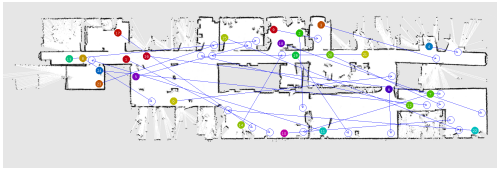
Empty Hall environment

The map of the *empty hall* environment and the endpoints of the well-formed infrastructure embedded in the environment are shown in Figure 4.6.1a. The roadmap used during trajectory planning is depicted

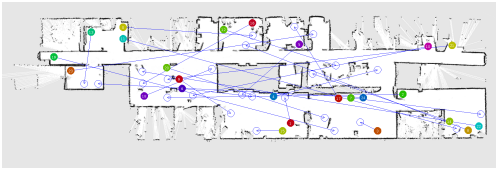
(a) *Office Corridor* Environment. The endpoints used to create a well-formed infrastructure in the environment are shown as red circles.



(b) A selected part of the roadmap used for trajectory planning.



(c) Example instance with free-formed tasks for 25 robots. Task of each robot shown in blue.



(d) Example instance with tasks for 25 robots in the well-formed infrastructure. Task of each robot shown in blue.

Figure 4.6.2: Office Corridor environment

in Figure 4.6.1b. For both instance sets and each number of robots ranging from $n = 1$ to $n = 50$, we generated 50 random instances of the problem containing the given number of robots.

Office Corridor environment

The map of the *office corridor* environment and the endpoints of the well-formed infrastructure embedded in the environment are shown in Figure 4.6.2a. The roadmap used during trajectory planning is depicted in Figure 4.6.2b. The map of the environment is based on the laser rangefinder log of the *Cartesium* building at the University of Bremen[1]. For both instance sets and each number of robots ranging from $n = 1$ to $n = 30$, we generated 50 random problem instances containing the given number of robots.

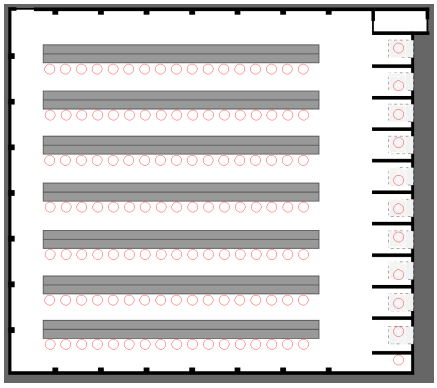
Warehouse environment

The map of the *warehouse* environment and the endpoints of the well-formed infrastructure embedded in the environment are shown in Figure 4.6.3a. The roadmap used during trajectory planning is depicted in Figure 4.6.3b. For both instance sets and each number of robots ranging from $n = 1$ to $n = 60$, we generated 50 random problem instances containing the given number of robots. The tasks in the well-formed infrastructure instance set represent a scenario of an automated logistic center where robots move goods between the gates and the storage shelves.

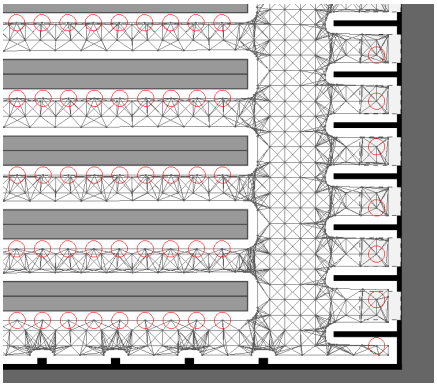
4.6.2 Experiment Setup

For each generated problem instance we run PP and RPP to obtain coordinated trajectories. Both PP and RPP were implemented to use an identical best trajectory planner. A best trajectory for each robot is computed with the time-extened roadmap planner (see Section 4.4), where the heuristic function is the time to travel along the shortest path on the roadmap from the given node to the goal node when dynamic obstacles are ignored. The experiments have been performed on a computer with AMD Opteron 8356 2.3GHz CPU and 8 GB RAM. For each algorithm we measure the following characteristics:

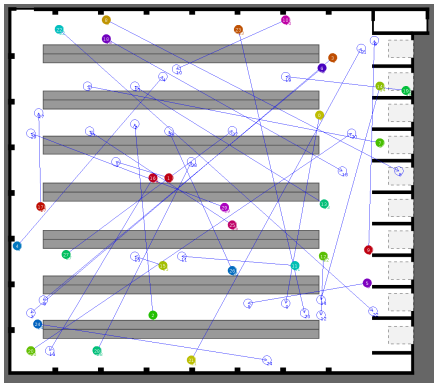
[1] We thank Cyrill Stachniss for providing the data through the Robotics Data Set Repository [Howard and Roy, 2003].



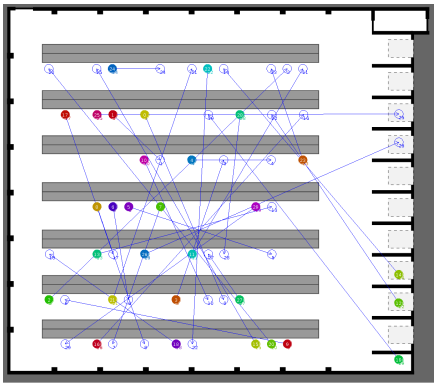
(a) *Warehouse* Environment. The endpoints used to create a well-formed infrastructure in the environment are shown as red circles.



(b) A selected part of the roadmap used for trajectory planning.



(c) Example instance with free-formed tasks for 30 robots. Task of each robot shown in blue.



(d) Example instance with tasks for 30 robots in the well-formed infrastructure. Task of each robot shown in blue.

Figure 4.6.3: Warehouse environment

- **Coverage:** We waited until each algorithm returns either a success or a failure and counted the number of instances each of the algorithms successfully solved.

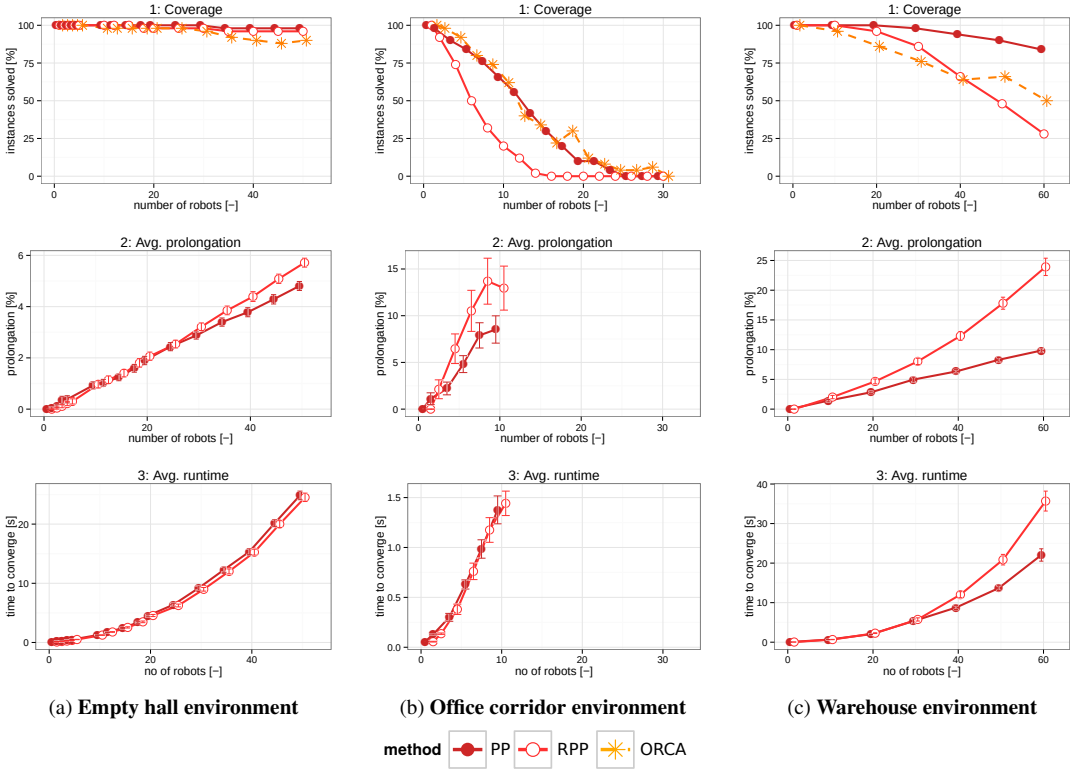
The following average characteristics were computed on all instances that were *successfully solved* by all compared algorithms. To obtain a reasonably precise estimate of the average, the values were computed only when there were at least 10 such instances for a particular number of robots.

- **Avg. runtime:** We measured the runtime that the algorithm needed to provide a collision-free solution.

- **Avg. prolongation:** The objective criterion we minimized is the sum of goal arrival times for each robot. We measured the duration of the trajectory for each robot and computed the prolongation coefficient for each instance as

$$\text{prolongation of alg. } A \text{ on instance } i = \frac{\sum_{i=1}^{n} t_i^A - t_i'}{\sum_{i=1}^{n} t_i'}, \tag{4.6.1}$$

  where t_i^A is the time the robot i needs to reach its goal position when it follows the trajectory



(a) **Empty hall environment** (b) **Office corridor environment** (c) **Warehouse environment**

Figure 4.6.4: Results: Free-formed Tasks (bars indicate standard error)

computed by the algorithm A and t'_i is the time the robot i would need to reach its goal following the shortest path on the roadmap if the collisions with other robots were ignored.

Comparison with reactive planning

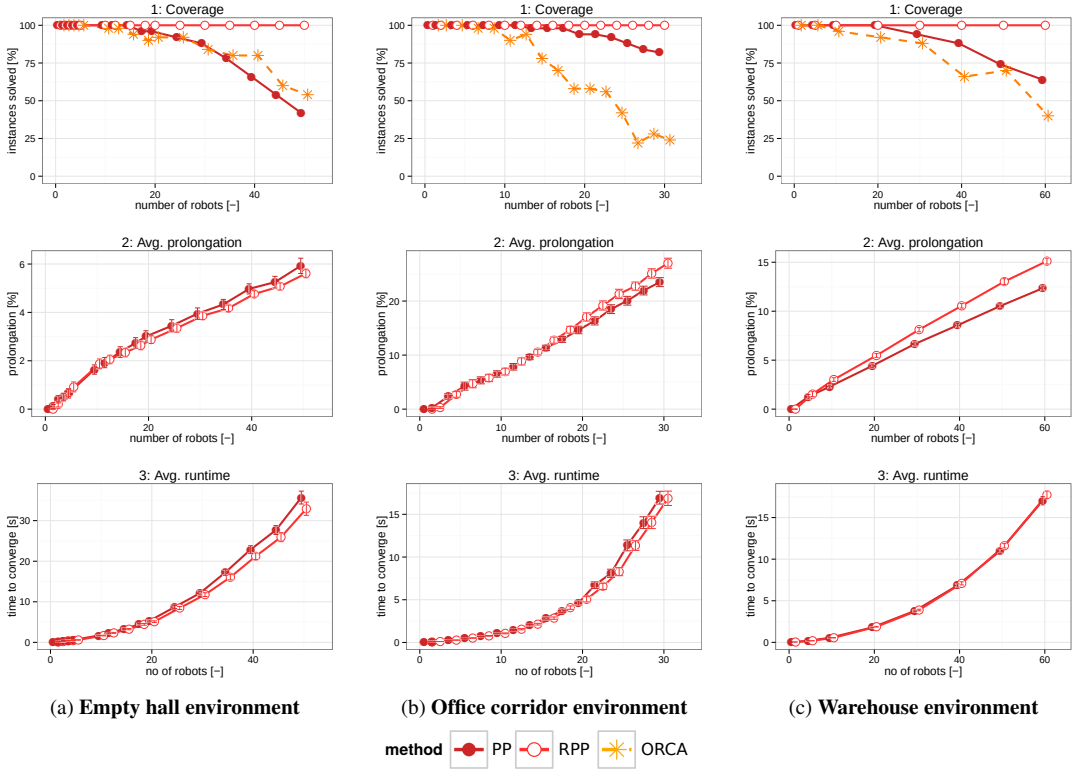
In order to evaluate the practical advantages of the proposed planning approach, we also compared its coverage against a popular reactive technique ORCA in our environments. When using ORCA, each robot continuously observes positions and velocities of other robots in its neighborhood. Should any potential collision be detected, a linear program is solved to obtain a new collision averting velocity that the robot should follow. If there are no imminent collisions, the robot follows its preferred velocity. In our implementation, the preferred velocity vector points at the shortest path from the robot's current position to the goal.

4.6.3 Results

The plots showing the results of the comparison on instances with free-formed tasks are in Figure 4.6.4. The results in the respective well-formed infrastructures are in Figure 4.6.5.

Coverage

On instances with free-formed tasks, all tested algorithms exhibit incomplete coverage of the instance space. Generally, RPP solved fewer instances than the PP due to the requirement on an $S^{>i}$-avoiding



(a) **Empty hall environment** (b) **Office corridor environment** (c) **Warehouse environment**

Figure 4.6.5: Results: Tasks in Well-formed Infrastructures (bars indicate standard error)

path used in RPP. Unlike PP, RPP requires that each robot follows a path that avoids regions occupied by lower-priority robots at their start position. Since the start positions for all robots were randomly generated, it can indeed happen that the generated start positions block some higher-priority robot from reaching its goal. For an example instance, where such a phenomena occurs consult Figure 4.3.3.

For tasks in well-formed infrastructures, the existence of an $S^{>i}$-avoiding path for each robot is guaranteed and thus the RPP algorithm shows full-instance coverage in accordance with our theoretical findings. Further, we can see (best in Figure 4.6.5a-1) that some of those instances remain unsolved both by PP and ORCA.

Note that in the Office Corridor and Warehouse environments with free-formed tasks, all tested algorithms exhibit low success rates on instances with high number or robots. That is, it is increasingly hard to randomly generate an instance on which all tested algorithms succeed. Therefore, the following aggregate characteristics are only computed if there is enough data from successfully solved instances.

Prolongation

In some of the environments (see, e.g., Figure 4.6.4c-2), we can observe that RPP generates longer trajectories than PP. This difference can be explained by RPP having to preemptively avoid start positions of the lower-priority robots, which in certain scenarios leads to significantly longer trajectories.

Runtime

In most environments, the runtime needed to return a solution by PP and RPP is comparable. However, in Warehouse environment with free-formed task (see Figure 4.6.4c-3), we can observe a significant difference between the average runtime of PP and RPP. The longer average runtime of RPP is caused by the trajectory planning routine having to explore a larger portion of the search space, since as we explained in the previous paragraph, the "best" trajectory obeying the constraints of RPP can be on average significantly longer than the "best" trajectory found by PP.

In this chapter, we focused on the centralized coordination in a multi-robot system using prioritized planning. We have analyzed when is the classical prioritized planning approach bound to fail and introduced a revised version of the prioritized planning algorithm that allowed us to state sufficient conditions for the polynomial solvability of a given instance by the revised algorithm. In particular, we have shown that if the operational environment is designed as a well-formed infrastructure, then RPP is guaranteed to return coordinated trajectories in cubic time in the number of coordinated robots. In the next chapter, we will discuss decentralized implementations of prioritized planning approach.

Chapter 5

Asynchronous Decentralized Prioritized Planning

In this chapter, we will discuss algorithms for the problem of decentralized batch coordination of multiple circular robots (Problem 8). Oftentimes, a decentralized algorithm for trajectory coordination is more desirable than a centralized one. Consider for example a large multi-robot system consisting of tens or hundreds of heterogeneous robots. One of the disadvantages of the centralized approach is that a central component has to gather a complete information about the current state, the constraints, and objectives of each robot in the system before it can compute a coordinated solution. With a decentralized algorithm, such information may remain private to each robot and the coordinated solution is found merely by exchanging a limited amount of shared information.

In particular, the algorithms discussed in this chapter compute solution by exchanging messages containing proposed future trajectories and each such trajectory is computed using a trajectory planner that runs on-board of the robot, which also creates an opportunity for a parallelization of the computation process. We generalize and simplify an existing synchronized decentralized method for trajectory coordination by removing the synchronization points in the algorithm, prove that the newly proposed asynchronous version is guaranteed to terminate, and experimentally show that the asynchronicity of the algorithm improves the speed of its convergence. Further, we show that the both decentralized implementations of RPP algorithm inherit the properties of its centralized counterpart and therefore they can be used for guaranteed multi-robot coordination in well-formed infrastructures.

5.1 Synchronized Decentralized Prioritized Planning

A decentralized implementation of the classical prioritized planning scheme, where robots concurrently proceed in synchronized rounds, has been first presented by Velagapudi et al. [2010]. We use their approach as a baseline decentralized implementation of both classical and revised prioritized planning and denote the resulting algorithm as *synchronized decentralized implementation of prioritized planning* (SD-PP) and *synchronized decentralized implementation of revised prioritized planning (SD-RPP)*. The SD-(R)PP abbreviation is used when a statement holds for both variants.

The algorithm proceeds in synchronized rounds. In every round, each robot ensures that its current trajectory is consistent with the trajectories of higher-priority robots from the previous round. If the current trajectory is consistent, then the robot keeps its current trajectory and remains silent. Otherwise, it finds a new consistent trajectory for itself and broadcasts the trajectory to all other robots. When a robot finishes its computation in the current round, it waits for all other robots to finish the round and all robots simultaneously proceed to the next round. The algorithm successfully terminates if none of the robots changes its current trajectory during a single round. The SD-PP algorithm terminates with

failure if there is a robot that fails to find a trajectory that avoids the higher-priority robots following their respective trajectories. The SD-RPP algorithm, on the other hand, finishes with failure if there is a robot that fails to find a satisfying trajectory that avoids the start positions of lower-priority robots. The pseudocode of SD-(R)PP is listed in Algorithm 7.

Algorithm 7: Synchronized Decentralized Implementation of (Revised) Prioritized Planning. Pseudocode for robot i

Algorithm `SD-(R)PP`
  $\pi_i \leftarrow \emptyset$;
  $H_i \leftarrow \emptyset$;
  $S \leftarrow \begin{cases} \emptyset & \text{for SD-PP} \\ \bigcup_{j>i} S^j & \text{for SD-RPP} \end{cases}$;
  repeat
    $\pi^* \leftarrow$ `Find-consistent` $(\pi_i, \mathcal{W} \setminus S, \Delta(H_i))$;
    if $\pi^* = \emptyset$ **then**
      report failure and terminate;
    else if $\pi^* \neq \pi_i$ **then**
      $\pi_i \leftarrow \pi^*$;
      broadcast $\mathrm{INFORM}(i, R_i^{\Delta}(\pi^*))$;
    wait for INFORM messages from all other robots, wait for all other robots to finish processing INFORM messages ;
  until *not global termination detected*;
Handle-message $\mathrm{INFORM}(j, \Delta_j)$
  if $j < i$ **then**
    $H_i \leftarrow \left(H_i \setminus \{(j, \Delta_j') \,:\, (j, \Delta_j') \in H_i\}\right) \cup \{(j, \Delta_j)\}$;
Function `Find-consistent` $(\pi, \mathcal{W}, \Delta)$
  if $\pi = \emptyset \vee \neg\mathrm{consistent}_i(\pi, \Delta)$ **then**
    $\pi^* \leftarrow$ `Best-traj`$_i$ $(\mathcal{W}, \Delta)$;
    return π^*;
  else
    return π;

In SD-(R)PP, each robot i maintains a database of space-time regions occupied by higher-priority robots. We call such a database a trajectory store and model it as a set of pairs $H_i = \{(j, \Delta_j)\}$, where Δ_j is the space-time region occupied by robot j.

Function $\Delta(H)$ represents the region of the space-time occupied by all robots stored in a trajectory store H:

$$\Delta(H) := \bigcup_{(j,\Delta_j)\in H} \Delta_j.$$

Further, we use predicate $\mathrm{consistent_i}(\pi, \Delta)$ to express that the trajectory π of robot i is collision-free against dynamic obstacles Δ, defined as

$$\mathrm{consistent}_i(\pi, \Delta) := R_i^{\Delta}(\pi) \cap \Delta = \emptyset.$$

Properties

In order to facilitate and simplify the exposition and analysis of the later introduced asynchronous algorithm, we developed an alternative proof of termination of the SD-(R)PP algorithm, which deviates from the original one devised by Velagapudi et al. [2010]. Further, in this section we show that SD-(R)PP inherits the soundness and completeness properties from its respective centralized counterpart.

The SD-(R)PP algorithm is guaranteed to terminate. First, we need to define what termination means for a decentralized algorithm. A decentralized algorithm

- **terminates** when all robots stop computing,
- **terminates with a failure** if it terminates and there is at least one robot that reported a failure during the computation,
- **terminates successfully** if it terminates without a failure.

To show that SD-(R)PP terminates, we first show that robots running SD-(R)PP cannot exchange messages forever.

Proposition 18. *All robots running the SD-(R)PP algorithm eventually stop sending INFORM messages.*

Proof. We proceed by induction on the robot priority i.
Inductive hypothesis: Robots $1, \ldots, i-1$ eventually stop sending messages.
Base step (robot 1):
Robot 1 is the highest-priority robot and as such it does not receive any message from a higher-priority robot. Therefore, its trajectory store will stay empty. During the initialization, robot 1 either successfully finds its initial trajectory and broadcasts a single message, or reports a failure and terminates. Since its trajectory store is empty, its initial trajectory will never become inconsistent, the robot will therefore never replan and send any further INFORM messages.
Induction step (robot i):
From the inductive hypothesis, we have that each of the robots $1, \ldots, i-1$ eventually stops broadcasting messages. After the last message from robots $1, \ldots, i-1$ has been received by the robot i, its trajectory store gets updated for the last time since from our assumption there are no more messages from higher-priority robots. After the trajectory store changes for the last time, the robot either a) keeps its current trajectory if it is consistent with the last trajectory store, b) finds a new consistent trajectory or c) terminates with failure. In cases a) and b), the current trajectory will never become inconsistent again because the trajectory store does not change anymore and thus the robot will never have to replan and communicate a new trajectory. In the case c), the robot has terminated the computation and thus it will not send any more INFORM messages in the future. We see that after robots $1, \ldots, i-1$ stopped sending messages, also robot i eventually ceases to send messages. □

Corollary 19. *SD-(R)PP always terminates.*

Proof. We have that all robots in the system will eventually stop sending messages. Assume that the last message is broadcast during round k. In round $k+1$, no robot changes its trajectory since otherwise, a message would have to be broadcast which is a contradiction. If no robot changes its trajectory during the round, the global termination condition is satisfied and the system terminates. □

Unless a failure is reported by one of the robots, the solution computed when SD-(R)PP terminates is sound:

Proposition 20. *When SD-(R)PP successfully terminates, all robots hold trajectories that are mutually conflict-free.*

Proof. Let π_i be the trajectory of robot i after the algorithm has terminated. We need to show that

$$\forall i, j : \ i \neq j \Rightarrow \pi_i \text{ and } \pi_j \text{ are conflict-free.}$$

Take two arbitrary, but different robots i and j. Since the conflict-free relation is symmetrical, we can assume $j < i$ w.l.o.g. If the robot i stopped computing without a failure, it must have received the INFORM message from a higher-priority robot j carrying its last trajectory π_j at some point before its termination. Since there are no further INFORM messages broadcast by the robot j, the trajectory store of the robot i will contain π_j from that point on. Every trajectory returned by `Find-consistent` function for the robot i from that point on will be conflict-free with π_j and thus also its last trajectory π_i will be conflict-free with π_j. □

The synchronized decentralized implementations of PP and RPP inherit completeness properties from the respective centralized implementations. In general, both SD-PP and SD-RPP are incomplete. However if an $S^{>i}$-avoiding and $G^{<i}$-avoiding satisfying path exists for each robot, then the SD-RPP is guaranteed to terminate successfully.

Lemma 21. *If there is an $S^{>i}$-avoiding and $G^{<i}$-avoiding satisfying path for every robot i and a complete algorithm is used for the single-robot trajectory planning in* `Best-traj` *function, then SD-RPP is guaranteed to terminate with a conflict-free solution.*

Proof. The argument used in the proof of Theorem 12, which shows that RPP will never fail during planning, can be extended to decentralized implementations of RPP as follows: Take an arbitrary replanning request for robot i. All trajectories of each higher-priority robot $j < i$ have been generated to be $S^{>j}$-avoiding and thus such trajectory will be also S^i-avoiding. All trajectories in the trajectory store of the robot i are therefore S^i-avoiding. An $S^{>i}$-avoiding satisfying trajectory consistent with trajectories of higher-priority robots can be constructed as follows: Wait at the starting position s_i until all higher-priority robots reach their goal position and then follow the $S^{>i}$-avoiding $G^{<i}$-avoiding satisfying path from the assumption. Since such a trajectory is guaranteed to exist for robot i and a complete replanning algorithm is used, the replanning cannot report failure. The algorithm must terminate with success. □

5.2 Asynchronous Decentralized of Prioritized Planning

Due to its synchronous nature, the SD-(R)PP algorithm does not fully exploit the computational resources distributed among individual robots. In every iteration, the robots that finished their trajectory planning routine earlier, or did not have to re-plan at all, stay idle while waiting for the slower computing robots in that round. However, they could use the time to resolve some of the conflicts among themselves and speed up the overall process. An example of a situation, where the asynchronous algorithm would be beneficial is illustrated in Figure 5.2.1.

To deal with such an inefficiency, we propose an asynchronous decentralized implementation of classical prioritized planning, abbreviated AD-PP, and revised prioritized planning scheme, abbreviated AD-RPP. The AD-(R)PP abbreviation is used when a statement holds for both variants.

The pseudocode of AD-(R)PP is shown in Algorithm 8. The asynchronous algorithm replaces the concept of globally synchronized rounds (while loop in Algorithm 7) by a reactive approach in which every robot reacts merely to incoming INFORM messages. Upon receiving an INFORM message (**Handle-message** $\mathrm{INFORM}(j, \Delta_j)$ routine in Algorithm 8), the robot simply replaces the information about the trajectory of the sending robot in its trajectory store and checks whether its current trajectory is still consistent with the new contents of its trajectory store. If the current trajectory is inconsistent, the robot triggers replanning and informs other robots about its new trajectory. Otherwise, the robot keeps its current trajectory and remains silent.

Algorithm 8: Asynchronous Decentralized Implementation of (Revised) Prioritized Planning

Algorithm `AD-(R)PP`
  $\pi_i \leftarrow \emptyset$;
  $H_i \leftarrow \emptyset$;
  $S \leftarrow \begin{cases} \emptyset & \text{for AD-PP} \\ \bigcup_{j>i} S^j & \text{for AD-RPP} \end{cases}$;
  $\pi_i \leftarrow$ `Find-consistent` $(\pi_i, \mathcal{W} \backslash S, \Delta(H_i))$;
  if $\pi_i = \emptyset$ **then**
    report failure and terminate;
  else
    broadcast INFORM$(i, R_i^{\Delta}(\pi_i))$;
    wait for global termination;
Handle-message INFORM(j, Δ_j)
  if $j < i$ **then**
    $H_i \leftarrow \left(H_i \setminus \{(j, \Delta'_j) : (j, \Delta'_j) \in H_i\}\right) \cup \{(j, \Delta_j)\}$;
    $\pi^* \leftarrow$ `Find-consistent` $(\pi_i, \mathcal{W} \setminus S, \Delta(H_i))$;
    if $\pi^* = \emptyset$ **then**
      report failure and terminate;
    else if $\pi^* \neq \pi_i$ **then**
      $\pi_i \leftarrow \pi^*$;
      broadcast INFORM$(i, R_i^{\Delta}(\pi^*))$;

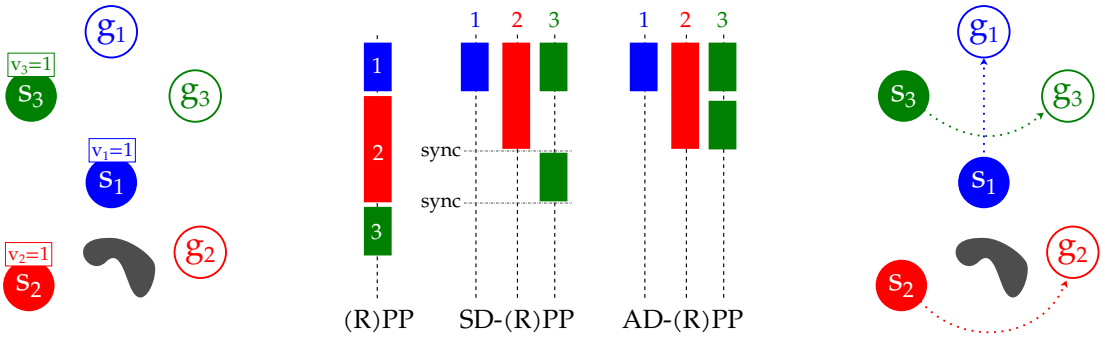


Figure 5.2.1: Example problem in which AD-(R)PP converges faster than SD-(R)PP. **Left:** The task of robots 1, 2, and 3. The robot i travels from $\mathbf{s}_i$ to $\mathbf{g}_i$. The gray area represents an obstacle. **Middle:** Sequence diagrams showing the planning process in (R)PP, SD-(R)PP and AD-(R)PP. Suppose that robot 2 needs to plan longer, because it has to plan around the gray obstacle. In SD-(R)PP, the robot 3 starts resolving the conflict with robot 1 only when robot 2 has finished computing its trajectory in the first round. In AD-(R)PP, robot 3 stars resolving the conflict with robot 1 immediately after it becomes aware of it, therefore it finds the solution faster. **Right:** The final solution.

Properties

AD-(R)PP inherits all the desirable properties from its synchronized and centralized counterparts, i.e., it terminates and if it terminates with success, then all the robots will hold conflict-free trajectories. Further, AD-RPP is guaranteed to solve instances that admit an $S^{>i}$-avoiding and $G^{<i}$-avoiding path for each robot.

Proposition 22. *AD-(R)PP always terminates.*

Proof. Recall that the inductive argument demonstrating that robots running SD-(R)PP will eventually stop sending messages (Lemma 18) does not make use of the synchronization points in SD-(R)PP and thus it is also valid for AD-(R)PP. By this argument, we know that there is a finite number of messages being sent. We can observe that a robot running AD-(R)PP performs computation only during initialization or when it processes an incoming message. When all robots process their last incoming messages, the system terminates. □

Proposition 23. *When AD-(R)PP successfully terminates, all robots hold trajectories that are mutually conflict-free.*

Proof. The proof of the soundness of SD-(R)PP (Proposition 20) is directly applicable also to AD-(R)PP. □

Proposition 24. *If an $S^{>i}$-avoiding, $G^{<i}$-avoiding satisfying path exists for each robot i and a complete algorithm is used for the single-robot trajectory planning in* `Best-traj` *function, then AD-RPP terminates.*

Proof. The proof of Proposition 21, where this property is demonstrated to hold for SD-RPP, is directly applicable also for AD-RPP. □

5.3 Complexity

In order to derive computational complexity of SD-(R)PP and AD-(R)PP algorithms, we will assume that `Best-traj` routine is implemented using the time-extended roadmap planner as described in Section 4.4 on a given spatial roadmap $G = (V, E)$.

Recall that the number of robots is denoted n. Further, the number of vertices in roadmap G is denoted as v, the length of the longest edge in the roadmap G is denoted as d, the maximum speed of the robot is $v_{\max}$, the step of time discretization in time-extended roadmap is δ, and k is the number of time steps of the longest edge in time-extended roadmap computed as $k := \left\lceil \frac{d}{v_{\max} \cdot \delta} \right\rceil$. Then, we can show that the worst-case computational complexity of a single replanning in SD-(R)PP and AD-(R)PP algorithms is quadratic in the number of higher-priority robots in the system:

Lemma 25. *Function* `Find-consistent` *executed by robot i in SD-(RPP) and AD-(R)PP will return a collision-free trajectory π if it exists or reports a failure to find one in time $O(i^2k^2v^4)$ and the trajectory π will have bounded cost $\hat{c}^{\text{arr}}(\pi, \mathbf{g}_i) \leq ik\delta v$.*

Proof. We will proceed by induction on robot priorities.

Inductive assumption: The trajectory π_j of each robot $j = 1, \ldots, i$ at any point of the computation has bounded cost $\hat{c^{\text{arr}}}(\pi_j, \mathbf{g}_j) \leq ik\delta v$ and consequently $\forall t \geq ik\delta v : \ \pi_j(t) = \mathbf{g}_j$.

Base case: There are no robots with higher priority than the robot 1. Therefore, the robot 1 never receives any INFROM message and the set of dynamic obstacles Δ is always empty and consequently, it trivially stops changing at time $t' = 0$. From Lemma 15 we can get the bound on the cost of trajectory for robot 1 denoted π_1 as $\hat{c}^{\text{arr}}(\pi_1, \mathbf{g}_1) \leq \max(0, 0) + k\delta v \Rightarrow \hat{c}^{\text{arr}}(\pi_1, \mathbf{g}_1) \leq k\delta v$. From Lemma 16 and

substituting $\tau = \max(0,0)/\delta + kv = kv$, we know that the trajectory π_1, if it exists, will be found in time $O(k^2v^4)$.

Inductive step (i-th robot): The dynamic obstacle region Δ is be composed only of space-time regions occupied by robots $1,\dots,i-1$ received through INFROM messages. From the inductive assumption, we know that the trajectory of any robot $1,\dots,i-1$ arrives to its goal before time $(i-1)k\delta v$. Thus, the set of dynamic obstacles Δ stops changing at $t' \leq (i-1)k\delta v$. From Lemma 15 we can get bound on the cost of trajectory π_i as

$$\begin{aligned}
\hat{c}^{\mathrm{arr}}(\pi_i, \mathbf{g}_i) &\leq \max(0, t') + k\delta v \\
\hat{c}^{\mathrm{arr}}(\pi_i, \mathbf{g}_i) &\leq t' + k\delta v \\
\hat{c}^{\mathrm{arr}}(\pi_i, \mathbf{g}_i) &\leq (i-1)k\delta v + k\delta v \\
\hat{c}^{\mathrm{arr}}(\pi_i, \mathbf{g}_i) &\leq ik\delta v,
\end{aligned}$$

and subsequently the goal arrival time t^{arr} for trajectory π_i is bounded by $ik\delta v$. From Lemma 16 and substituting $\tau = \max(0,t')/\delta + kv = t' + kv = (i-1)kv + kv = ikv$, we know that the trajectory π_1, if it exists, will be found in time $O(i^2k^2v^3 + i^2k^2v^4) = O(i^2k^2v^4)$.

Next, we derive a theoretical bound on the maximum number of messages broadcast by each robot and on the maximum number of replannings performed by each robot. □

Lemma 26. *When running SD-(R)PP, a single robot broadcasts at most n messages and performs at most n replannings.*

Proof. SD-(R)PP algorithm runs in synchronized rounds. We will proceed by induction on algorithm iterations. **Inductive assumption:** After iteration i, the trajectories of robots $1,\dots,i$ are fixed.

Base case: At iteration 1, the robot 1 find its first trajectory. Since there are no higher-priority robots, its initial trajectory is trivially consistent and therefore fixed also in every subsequent iteration.

Inductive case: At iteration i, robot either has a trajectory that is already collision-free with respect to the robots $1,\dots,i-1$ or will find one. From our assumption, the trajectories of robots $1,\dots,i-1$ do not change in iterations following the iteration i and thus the trajectory of robot i does not need to be replaned either. □

Since AD-(R)PP algorithm proceeds asynchronously, it provides worse theoretical bounds than SD-(R)PP.

Lemma 27. *When running AD-(R)PP, a single robot priority i may in the worst case perform 2^i replannings and broadcast up to 2^i messages.*

Proof. Let N_i denote the maximum number of messages broadcast by robot i and C_i the maximum total number of messages broadcast by robots $1,\dots,i$. We will proceed by induction on robot priorities.

Induction hypothesis: The maximum total number of messages broadcast by robots $1,\dots,i$ is $C_i = 2^i - 1$.

Base case: Since there are no higher-priority robots, robot 1 broadcasts only a single message with its initial trajectory. We have $N_1 = 1 = 2^0$ and $C_i = 1 = 2^1 - 1$.

Inductive case: Robot i broadcasts one message with its initial trajectory and then it may be forced to replan after each message from a higher-priority robot. Therefore, $N_i = C_{i-1} + 1$ and $C_i = C_{i-1} +$

N_i, which can be rearranged as

$$\begin{aligned} C_i &= C_{i-1} + N_i \\ &= C_{i-1} + C_{i-1} + 1 \\ &= 2C_{i-1} + 1 \\ &= 2(2(\ldots 2(2 \cdot 1 + 1) + 1 \ldots) + 1) + 1 \\ &= 2 \cdot 2 \cdot \ldots \cdot 2 \cdot 2 + 2^{i-2} + 2^{i-3} + \ldots + 2 + 1 \\ &= 2^{i-1} + 2^{i-2} + \ldots + 2 + 1 \\ &= 2^i - 1. \end{aligned}$$

Then, $N_i = C_{i-1} + 1 = 2^{i-1} + 1 - 1 = 2^{i-1}$. □

Corollary 28. *When running SD-(R)PP, then the worst-case total computation time of a single robot is* $O(n^3k^2v^4)$.

Proof. Follows from Lemma 25 and Lemma 26. □

Corollary 29. *When running AD-(R)PP, then the worst-case total computation time of a single robot is* $O(2^nk^2v^4)$.

Proof. Follows from Lemma 25 and Lemma 27. □

Despite the catastrophic theoretical bound we derived on the number of messages broadcast by the asynchronous algorithm, the number of messages broadcast by the synchronous and the asynchronous algorithm is in practice similar. More importantly, as we show in the following section, the number of messages broadcast by each algorithm is small. In our experiments, an average robot broadcasts less than four messages during the resolution process.

5.4 Experimental Evaluation

In this section we report the results of empirical comparison between algorithms discussed in the previous two chapters (i.e., PP, RPP, SD-PP, SD-RPP, AD-PP, and AD-RPP) using multi-robot simulation in three real-world environments. The *effectiveness* of the algorithms is compared by a) measuring their ability to successfully solve a set of randomly generated problem instances (i.e., instanceset coverage) in three real-world environments and b) measuring the quality of the returned solutions. Their *efficiency* is compared a) by measuring the wall-clock time each algorithm requires to compute a valid solution and b) by counting the number of messages communicated by each algorithm.

5.4.1 Environments

The experiments were performed in three real-world environments: *Empty hall*, *Office corridor* and *Warehouse*. For each of the environments we generated two sets of problem instances: a) In the *free-formed tasks* instanceset, each robot is assigned a task to move from a randomly selected start position to a randomly selected goal position. b) In the *tasks in well-formed infrastructure* instanceset, we generated a set of endpoints that together with a particular roadmap discretization of the environment form a well-formed infrastructure; each robot is then assigned a random endpoint as a start position and a randomly chosen endpoint as a destination position. For the detailed description of the environments used in the experimental evaluation, see Section 4.6.1.

5.4.2 Experiment Setup

For each problem instance, we create a simulated multi-robot team and let all the robots coordinate the trajectories from the given start positions to the given goal positions using each of the tested algorithms. For the centralized algorithms (PP and RPP), the robots communicate their start position and goal position to a central solver that uses the information to compute a coordinated solution on 1 CPU and consequently sends a message with the resulting trajectory to each robot. For the decentralized algorithms (SD-PP, SD-RPP, AD-PP and AD-RPP), we assume that each robot uses its own on-board CPU to compute its trajectory. To measure the runtime characteristics of the execution of decentralized algorithms, we emulate the concurrent execution of the algorithms using a discrete-event simulation. The simulation measures the execution time of each message handling and uses the information to simulate the concurrent execution of the decentralized algorithm as if it were executed on n independent CPUs, where n is the number of robots. The individual robots communicate via an idealized simulated communication channel modeled as a perfectly reliable channel with zero latency. All compared algorithms use an identical best trajectory planner. The best trajectory for each robot is computed with the time-extended roadmap planner (see Section 4.4), where the heuristic function is the time to travel along the shortest path on the roadmap from the given node to the goal node when dynamic obstacles are ignored. The simulation of the decentralized system was implemented using the Alite multi-agent simulation toolkit. The test instances, the simulator, and Java implementation of individual algorithms can be downloaded from `https://github.com/mcapino/adpp-journal`. A video demonstrating the performance of ADPP and ADRPP algorithms on six selected instances is available at `https://youtu.be/dFm-JJhyuv0`. The experiments have been performed on a computer with AMD Opteron 8356 2.3GHz CPU and 8 GB RAM. For each algorithm we measure the following characteristics:

- **Coverage:** We waited until each algorithm returns either a success or a failure and counted the number of instances each of the algorithm successfully solved.

The following average characteristics were computed on all instances that were *successfully solved* by all compared algorithms. To obtain a reasonably precise estimate of the average, the values were computed only when there were at least 10 such instances for a particular number of robots.

- **Avg. time to solution:** We measured wall-clock runtime needed to compute a solution. For the centralized planner we recorded the time of termination of the centralized planner. For the decentralized algorithms, we recorded the time when the last robot detected global termination of the computation.

- **Avg. speed-up:** In order to be able to judge the effect of asynchronous execution of AD-(R)PP algorithm, we also computed the speed-up ratio for both decentralized algorithms over their centralized counterparts. The speed-up for the algorithm A on an instance i is computed as

  $$\frac{\text{runtime of centralized variant of alg. } A\text{on instance } i}{\text{runtime of alg. } A \text{ on instance } i},$$

  where the centralized variant of AD-PP and SD-PP is PP, and the centralized variant of AD-RPP and SD-RPP is RPP.

- **Avg. messages sent:** Every time a robot running a decentralized algorithm adopts a new trajectory (replans), the trajectory is broadcast to all other robots. Therefore the number of INFORM messages sent by a robot directly corresponds to the number of replannings performed by the robot.

- **Avg. prolongation:** The objective criterion we minimized is the sum of goal arrival times for each robot. We measured the duration of the trajectory for each robot and computed the prolongation

coefficient for each instance as

$$\text{prolongation of alg. } A \text{ on instance } i = \frac{\sum_{i=1}^{n} t_i^A - t'_i}{\sum_{i=1}^{n} t'_i}, \tag{5.4.1}$$

where t_i^A is the time the robot i needs to reach its goal position when it follows the trajectory computed by the algorithm A and t'_i is the time the robot i would need to reach its goal following the shortest path on the roadmap if the collisions with other robots were ignored.

Comparison with reactive planning

Similarly to the experimental comparison of centralized approaches, we also attempted to solve each of the instances using a reactive approach ORCA. For details on how was ORCA implemented, see Section 4.6.2.

5.4.3 Results

The plots showing the results of the comparison on instances with free-formed tasks are in Figure 5.4.1, the results in the respective well-formed infrastructures are in Figure 5.4.2.

Coverage

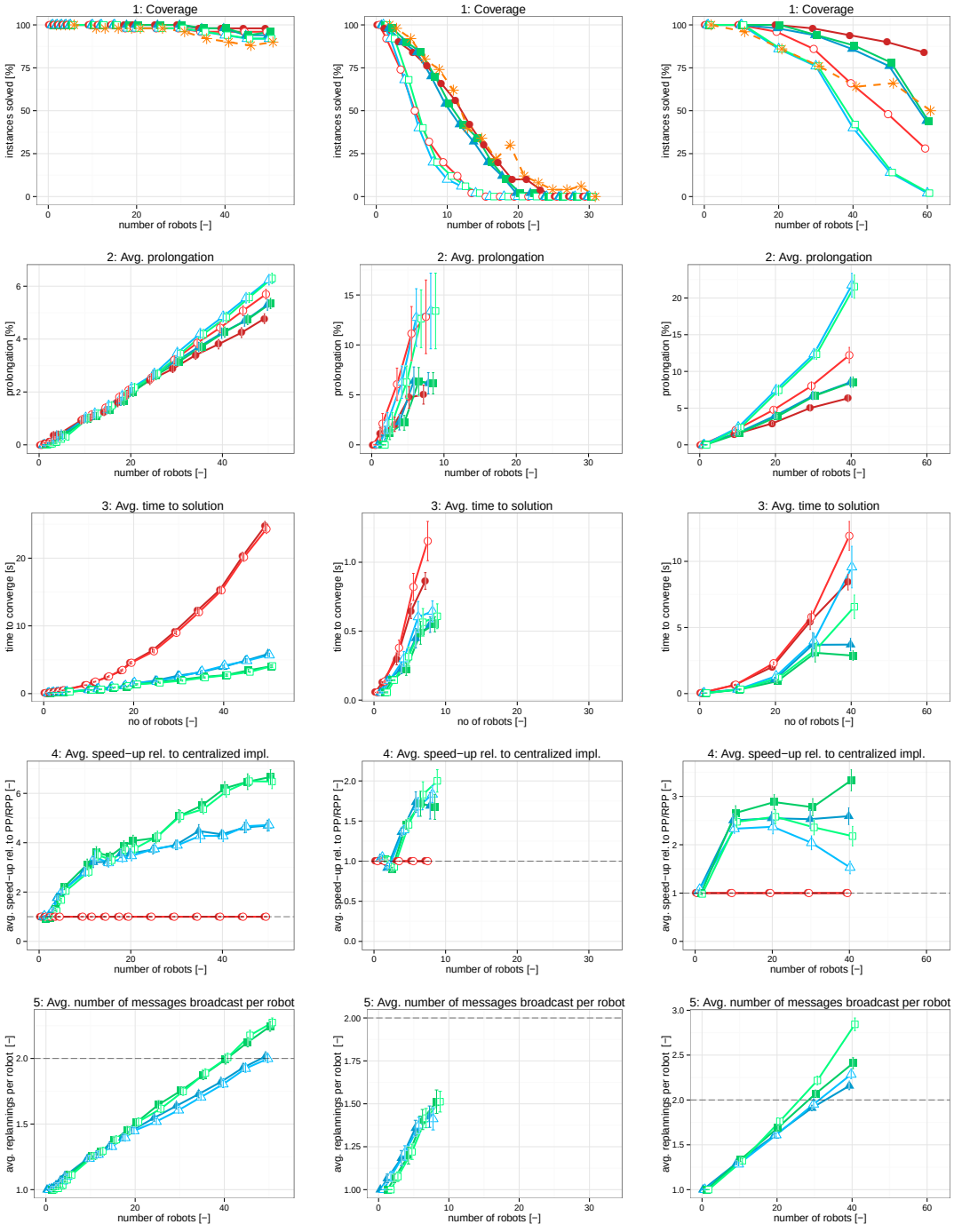
Consistently with what we observed in the comparison of centralized algorithms (see Section 4.6.3), on the free-formed instances the RPP-based algorithms cover smaller part of the instance space than the PP-based algorithms. However, on the instances in well-formed infrastructures, the RPP-based algorithms show full instance coverage and solve even those instances, where both PP-based algorithms and ORCA fail.

Further, we can see that the on free-formed instances, the decentralized algorithms exhibit smaller coverage compared to their centralized counterparts. The effect is negligible in Empty hall and Office corridor, but significant in Warehouse environment. The lower success rate of decentralized algorithms can be explained as follows: When a solution is computed using PP or RPP, the trajectory for each robot is computed only once. If no satisfying collision-free trajectory is found at this step, the process terminates with failure. On the contrary, decentralized algorithms may opportunistically replan the trajectory of each robot a number of times while only the last motion plan is used in the final solution. If any of the opportunistic replannnings fails because there is no trajectory that is collision-free with respect to current trajectories of the higher-priority robots, the planning process terminates with failure altogether even though it would be able to find a collision-free trajectory in future once some of the higher-priority robots change their trajectory. Since the decentralized algorithms have more "opportunities" to fail than the centralized algorithms, they exhibit lower success rate on the free-formed instances.

We note that in the Office Corridor and Warehouse environments with free-formed tasks, all tested algorithms exhibit low success rates on instances with a high number or robots. To obtain reasonably precise estimate of average, the following aggregate characteristics were only computed if there is at least ten successfully solved instances for a particular number of robots.

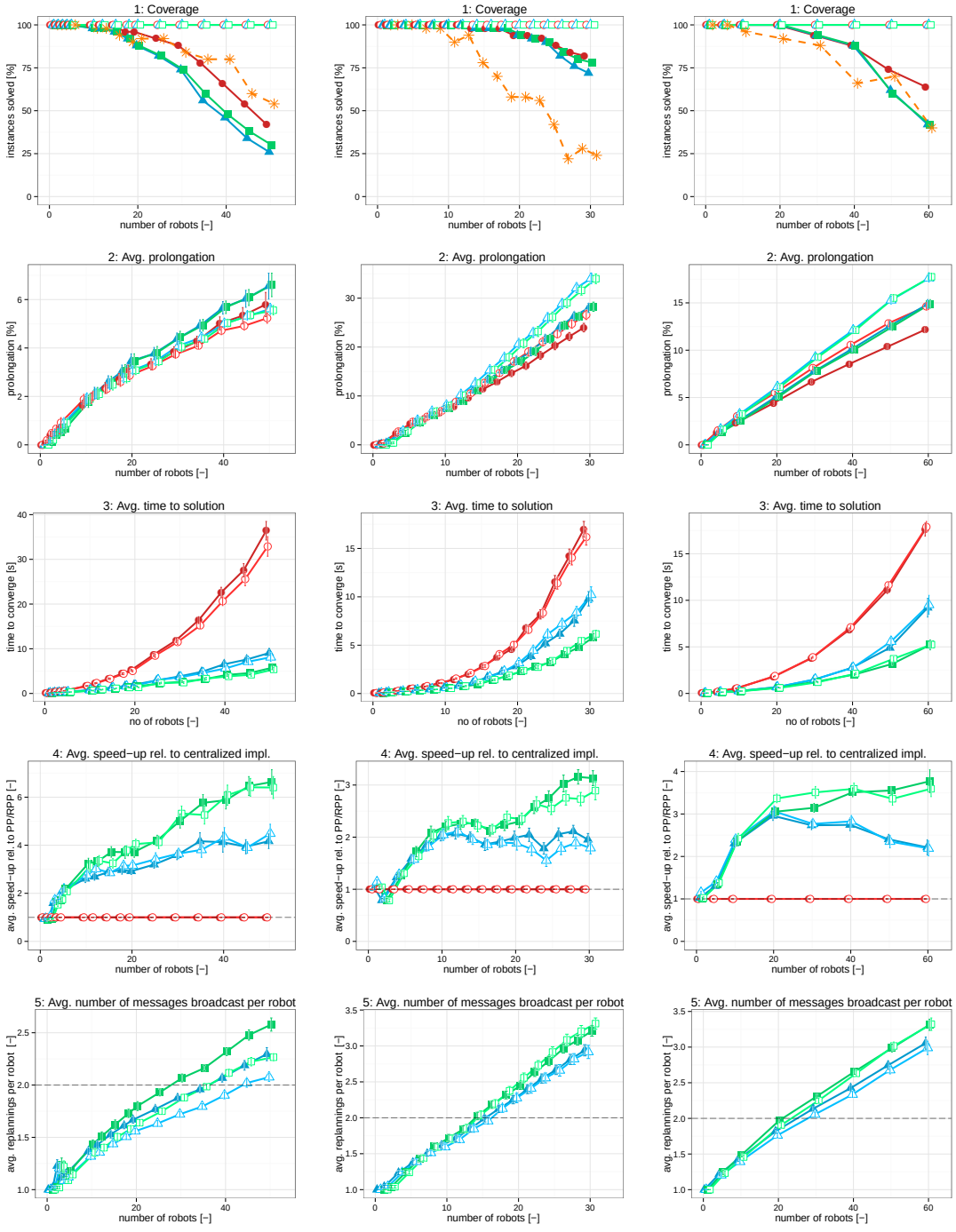
Time to solution/speed-up

The asynchronous decentralized implementation of both PP and RPP consistently achieves higher speed-up than the synchronized implementation in accordance with our prediction. The higher speed-up is exhibited on instances with higher number of robots, where it is more likely that several independent conflict clusters will occur. On such instances, it is often beneficial that the conflict clusters can resolve conflicts between the individual robots at different pace and thus converge faster. The phenomena can be seen clearly in Figures 5.4.1a-4, 5.4.2a-4, 5.4.2b-4, and 5.4.2a-4.



(a) **Empty hall environment** (b) **Office corridor environment** (c) **Warehouse environment**

Figure 5.4.1: Results: Free-formed Tasks
(bars indicate standard error)



(a) **Empty hall environment** (b) **Office corridor environment** (c) **Warehouse environment**

Figure 5.4.2: Results: Tasks in Well-formed Infrastructures
(bars indicate standard error)

Replannings/Communication

AD-(R)PP broadcasts a higher number of messages than SD-(R)PP. To see how this can be explained, suppose that at some point of the computation new conflicts arise between the trajectory of one particular robot and the trajectories of two other higher-priority robots. If the two conflicts occur in a single round, SD-(R)PP solves both conflicts during one replanning at the end of the round and therefore broadcasts only a single INFORM message. However, in such a situation AD-(R)PP may need to replan twice because it triggers replanning immediately after each of the conflicts is detected and thus it will broadcast two INFORM messages.

Prolongation

There are two phenomena influencing the quality of returned solutions. First, RPP-based algorithms generate slightly longer trajectories than PP-based algorithms. This is due to the fact that RPP preemptively avoids start positions of the lower-priority robots. Second, decentralized approaches generate slightly longer trajectories than the centralized approaches. The reason lies in the replanning condition used by the decentralized algorithms. The condition states that a robot should replan its trajectory only if the trajectory is inconsistent with the trajectories of other robots. Thus, the robot may receive an updated trajectory from a higher-priority robot that allows an improvement in its current trajectory, but since its current trajectory may be still consistent, the robot will not exploit such an opportunity for an improvement.

Sensitivity of results

The experiments were performed using a multi-robot simulation that uses an idealized model of communication and trajectory tracking. The first issue is that in our model the message exchange among the robots is assumed to be reliable and instant. In the context of indoor multi-robot systems, reliable communication can be realized by covering the entire workspace by a wireless LAN network and ensuring that a reliable protocol on MAC or transport layer (e.g., TCP) is used. In other contexts, e.g., when robots communicate using ad-hoc peer-to-peer communication, such a reliable wireless network might not be available and the robots will face communication unavailability or message loss. In practice, the ADPP algorithm can be altered to fit such systems by making each robot to periodically broadcast its current trajectory as discussed in Section 9.3. Due to the asynchronous stateless nature of the ADPP protocol, the algorithm is able to recover from the message loss and resolve conflicts with another robot immediately after at least one of the periodically broadcast messages from that robot is successfully delivered.

Further, real-world communication has latency. If the latency cannot be neglected relative to the time required to replan a trajectory, one should expect a lower speed-up ratio than the one reported by our simulation.

The second issue is that real robots suffer from imprecise sensing and actuation, which leads to errors in localization and trajectory tracking. This problem must be addressed by all open-loop planning algorithms and can to a certain extent be resolved by accounting for such uncertainties during the planning. A simple approach is to consider a sufficiently large buffer around the body of the robot that will empirically embed the uncertainty in localization and tracking.

In this chapter, we have proposed a novel decentralized algorithm for trajectory coordination in multi-robot teams. We have shown that the algorithm is guaranteed to terminate and inherits the completeness properties from its centralized counterpart. Compared to the existing synchronized decentralized algorithm, the asynchronous version is easier to implement and our experiments show that it exploits better the distributed computational power in the multi-robot team, leading to up to 2x faster convergence. In the next chapter, we will propose an adaptation of the prioritized planning approach for online coordination in a multi-robot system.

Chapter 6

Online Trajectory Coordination

In this chapter, we extend the prioritized planning approach to systems where the robots are not coordinated in batch, but rather incrementally during each task assignment, i.e., we will focus on the problem of online multi-robot coordination in infrastructures (Problem 10). To see when can be such an algorithm useful, consider a factory where the intermediate products are moved between the individual workstations by automated transport robots. A worker in such a factory can call a robot to its workstation, puts an item in a basket mounted on the robot and orders the robot to relocate to another workstation. To successfully fulfill the task, the robot has to coordinate its motion with other transport robots operating in the factory. Even though the coordinated trajectories can be found using a batch multi-robot trajectory planner from the current positions of the robots to their destinations, the other robots may be at the moment of planning outside of an endpoint and thus RPP, SD-RPP, and AD-RPP in this setup cannot guarantee the ability to find a solution. In this chapter, we will introduce an algorithm called Continuous Best Response Approach that achieves guaranteed performance even if the tasks are assigned to robots incrementally. We show that if the robots operate in a well-formed infrastructure, then a collision-free trajectory for any task between the endpoints of the infrastructure is guaranteed to exist and moreover, it will be computed in quadratic time in the number of robots in the system.

6.1 COBRA – General Scheme

In this section, we will introduce Continuous Best-Response Approach (COBRA), a decentralized method for trajectory coordination in multi-robot systems with incrementally assigned tasks. In the general formulation of the algorithm, we assume that each of the robots in the system is able to compute an optimal trajectory for itself from its current position to a given destination position in the presence of moving obstacles without prescribing how such a trajectory should be computed. In order to synchronize the information flow and ensure that the robots plan their trajectory using up-to-date information about the trajectories of others, robots use a distributed token-passing mechanism [Ghosh, 2010] in which the token is used as a synchronized shared memory holding current trajectories of all robots. We identify a token Φ with a set $\{(a_i, \pi_i)\}$, which contains at most one tuple for each robot $a = 1 \ldots, n$. Each such tuple represents the fact that robot a is moving along trajectory π. At any given time, the token can be held by only one of the robots and only this robot can read and change its content.

A robot that has been newly added to the system tries to obtain the token and to register itself with a trajectory that stays at its initial position forever. After all the robots have been added to the system, the user can start assigning relocation tasks to individual robots.

When a new relocation task is received by robot i, the robot requests the token Φ. When the token is obtained, the robot runs a trajectory planner to find a new "best-response" trajectory to fulfill the relocation task. The trajectory is required a) to start at the robot's current position p at time $t_{\text{now}} +$

Algorithm 9: COBRA – specification for robot i. The current time is denoted t_{now}, the maximum time that can be spent in trajectory planning is denoted t_{planning}.

On *registered to the system at position* $\mathbf{s}$
  $\Phi \leftarrow$ request token ;
  $\pi \leftarrow \pi(t)$ such that $\forall t \in [0, \infty) : \ \pi(t) = \mathbf{s}$;
  $\Phi \leftarrow \Phi \cup \{(i, \pi)\}$;
  release token Φ ;
On *new relocation task* $\mathbf{s} \rightarrow \mathbf{g}$ *assigned*
  $\Phi \leftarrow$ request token ;
  assert $\mathbf{g}$ is not a destination of another robot;
  $\Phi \leftarrow (\Phi \setminus \{(i, \pi') \ : \ (i, \pi') \in \Phi\})$;
  $\Delta \leftarrow \bigcup_{(j,\pi_j)\in\Phi} R_j^{\Delta}(\pi_j)$;
  $t_{\text{dep}} \leftarrow t_{\text{now}} + t_{\text{planning}}$;
  $\pi \leftarrow$ `Best-traj`$_i$ ($\mathbf{s}$,t_{dep},$\mathbf{g}$,Δ) ;
  if $\pi = \emptyset$ **then**
    report failure
  $\Phi \leftarrow \Phi \cup \{(i, \pi)\}$;
  release token Φ ;
  start following π at t_{dep} ;
Function `Best-traj`$_i$ ($\mathbf{s}$,t_s,$\mathbf{g}$,Δ)
  return trajectory π for robot i that reaches $\mathbf{g}$ in minimal time such that
    a) $\pi(t_s) = \mathbf{s}$,
    b) $\exists t_g \, \forall t' \in [t_g, \infty) : \ \pi(t') = \mathbf{g}$,
    c) $R_i^{\Delta}(\pi) \cap \Delta = 0$
  if it exists, otherwise return $\emptyset$;

t_{planning} (at the end of the planning window), b) to reach the goal position $\mathbf{g}$ as soon as possible and remain at $\mathbf{g}$ and c) to avoid collisions with all other robots following trajectories specified in the token. If such a trajectory is successfully found, the token is updated with the newly generated trajectory and released so that other robots can acquire it. Then, the robot starts following the found trajectory. Once the robot successfully reaches the destination, it can accept new relocation tasks. The pseudocode of COBRA algorithm is listed in Algorithm 9.

Theoretical Analysis

In this section, we will derive a sufficient condition under which is the presented mechanism complete, i.e., it guarantees that all relocation tasks will be successfully carried out without any collisions. First, observe that, in general, a robot may fail to find a collision-free trajectory to its destination as illustrated earlier in Figure 2.2.2. In *well-formed infrastructures*, however, each trajectory planning is guaranteed to succeed and consequently, all relocation tasks will be carried out without collision.

Completeness in Well-formed Infrastructures

In this section, we show that if robots are operating in a well-formed infrastructure and use COBRA for trajectory coordination, they will successfully carry out all assigned relocation tasks without collisions. Our analysis assumes the following setup of the multi-robot system. First, the robots must operate in a well-formed infrastructure $(\mathcal{W}, E)$. When the system is initialized, each robot is located at a

distinct endpoint of the infrastructure. After the initialization is finished, robots start accepting relocation tasks from some higher-level component, which ensures that each destination is an endpoint of the infrastructure and two robots will never have simultaneously assigned relocation task with the same destination. Each robot uses a complete trajectory planner and any trajectory assigned to a robot will be precisely followed by the robot.

We start by defining two important properties of trajectories stored in a token: A token Φ is called a) *E-terminal* iff $\forall(a,\pi) \in \Phi : \pi$ is $\mathbf{g}$-terminal and $\mathbf{g} \in E$, and b) *collision-free* iff $\forall(a,\pi),\ (a',\pi') \in \Phi : a \neq a' \Rightarrow \pi$ and π' are collision-free. The following two propositions state that these two properties are maintained throughout the operation of the system, and consequently the system will be both deadlock-free and collision-free.

Proposition 30. *If token Φ acquired during handling of a new task $\mathbf{s} \to \mathbf{g}$ assigned to robot i (on line 7 in Algorithm 9) is E-terminal and collision-free, then the subsequent trajectory planning succeeds and returns a trajectory that is $\mathbf{g}$-terminal and collision-free with respect to other trajectories in Φ.*

Proof. Let $\mathrm{ext}_r\ X$ be an r-exterior of a set $X \in \mathbb{R}^2$ defined as $\mathrm{ext}_r\ X := \bigcup_{\mathbf{x}\in X} D(\mathbf{x}, r)$. Because $(\mathcal{W}, E)$ is a well-formed infrastructure and $\mathbf{s}, \mathbf{g} \in E$, there exists a $\mathrm{ext}_r\ (E \setminus \{\mathbf{s}, \mathbf{g}\})$-avoiding path p from $\mathbf{s}$ to $\mathbf{g}$. All trajectories in Φ are E-terminal, which implies that eventually they reach one of the endpoints and stay at that endpoint. Consequently, there exists a time point $\bar{t}$ after which all the robots reach their destination endpoints and stay there. A $\mathbf{g}$-terminal and collision-free trajectory $\pi : [t_s, \infty) \to \mathbb{R}^2$ for robot i can be constructed as follows:

- In time interval $t \in [t_s, \max(t_s, \bar{t})] : \pi(t) = \mathbf{s}$. The trajectory cannot be in collision with any other trajectory in Φ during this interval: From the assumption that new relocation tasks can be assigned only after the previous relocation task has been completed, we have $\forall t \in [t_{now}, \infty) : \pi(t) = \mathbf{g}'$. From the assumption that the start position of a new relocation task $\mathbf{s}$ must be the same as the goal of the robots previous task $\mathbf{g}'$, we know $\forall t > t_{now} : \pi(t) = \mathbf{s}$. Since Φ is collision-free, all trajectories of robots other than j from Φ must be avoiding $\mathbf{s}$ with $r_i + r_j$ clearance, where r_j is the radius of the other robot in Φ. Therefore, in time interval $[t_s, \max(t_s, \bar{t})]$, robot i will be collision-free with respect to other trajectories stored in Φ.

- In time interval $t \in [\bar{t}, \infty] : \pi$ follows path p until the goal position $\mathbf{g}$ is reached. The trajectory cannot be in collision with any trajectory in Φ during this interval: After time $\bar{t}$, all robots are at their respective goal positions G, which satisfy: a) $G \subseteq E$, b) $\mathbf{s} \notin G$, otherwise some of the trajectories would have to be in collision with π, which contradicts the finding from the previous point, and c) $\mathbf{g} \notin G$ because we assume that two robots cannot have simultaneously relocation tasks with the same destination. We know that the path p avoids regions $\mathrm{ext}_{2\overline{r}}\ (E \setminus \{\mathbf{s}, \mathbf{g}\})$ and thus the trajectory cannot be in collision with any other trajectory in Φ during the interval $[\bar{t}, \infty)$.

Such a trajectory can always be constructed and thus any single-robot complete trajectory planning method must find it. □

Proposition 31. *Every time Φ is acquired by a robot during new relocation task handling (line 7 in Algorithm 9), Φ is E-terminal and collision-free.*

Proof. By induction on Φ. Take an arbitrary sequence of individual robots acquiring the token Φ and updating it. Let the induction assumption be: Whenever Φ is acquired by a robot to handle a new relocation task, Φ is E-terminal and collision-free.

Base case: When the last robot registers itself, all robots have a trajectory in Φ that remains at their start positions forever. Since the start position are assumed to be distinct and at least $2\overline{r}$ apart, then Φ is collision-avoiding. Further, Φ is trivially E-terminal because after the last robot registers, all robots stay at their initial position, which is an endpoint.

Inductive step: From the inductive assumption, we know that Φ acquired at line 7 in Algorithm 1 is E-terminal and collision-free. Using Proposition 30, we see that for any relocation task $\mathbf{s} \to \mathbf{g}$, the algorithm will find a trajectory π that is g-terminal and collision-free with respect to other trajectories in Φ. From our assumption, the destination $\mathbf{g} \in E$. By adding trajectory π to Φ at line 15 in Algorithm 9, the token Φ remains E-terminal and collision-free. □

Theorem 32. *If a team of robots operates in a well-formed infrastructure $(\mathcal{W}, E)$ and the COBRA algorithm is used to coordinate relocation tasks between the endpoints of the infrastructure, then all relocation tasks will be successfully carried out without collision.*

Proof. By contradiction. Assume that a) there can be a relocation task $\mathbf{s} \to \mathbf{g}$ assigned to robot i that is not carried out successfully or b) two robot collide at some time point during the operation of the system.

Case A: A relocation task $\mathbf{s} \to \mathbf{g}$ assigned to a robot i has not been successfully completed. We assume that robots are able to perfectly follow any given trajectory π. Therefore either π is not $\mathbf{g}$-terminal or robot i collided with another robot during the execution of the trajectory. From Proposition 30 and 31 we know that the `Best-traj`$_i$ routine will always return a $\mathbf{g}$-satisfactory trajectory. The possibility that the robot collided while carrying out a relocation task is covered by Case B and is shown to be impossible. Thus, the relocation task $\mathbf{s} \to \mathbf{g}$ assigned to robot i will be completed successfully.

Case B: Robots i and j collide. Since we assume perfect execution of trajectories, it must be the case that there is π_i and π_j that are not collision-free. Since the collision relation is symmetrical, w.l.o.g., it can be assumed that π_j was added to Φ later than π_i. This implies that π_i was in the token Φ when π_j was generated within `Best-traj`$_j$ routine. However, this would imply that the trajectory planning returned a trajectory that is not collision-free with respect to trajectories in Φ, which is a contradiction with constraints used during the trajectory planning. Thus, robots i and j do not collide.

We can see that neither case A nor case B can be achieved and thus all relocation tasks are carried out without any collision. □

6.2 Complexity and Completeness on Roadmaps

In the previous section, we have presented the general scheme of COBRA that assumes that every robot is able to find an optimal best-response trajectory for itself using an arbitrary complete algorithm. In practice, such a planning would be often done via graph search in a discretized representation of the static workspace. In this section, we will analyze the properties of COBRA when all robots use a time-extended roadmap planner (TERP) described in Section 4.4 to plan their best-response trajectory.

The function `Best-traj`$_i$`(`$\mathbf{s}$,t_s,$\mathbf{g}$,Δ`)` used on line 12 in Algorithm 9 returns an optimal trajectory for a particular robot i from $\mathbf{s}$ to $\mathbf{g}$ starting at time t_s that avoids space-time regions Δ occupied by other robots. We will now assume that the robots use TERP to compute such a trajectory.

Completeness

In this section, we will show that COBRA with TERP operating on a roadmap for well-formed infrastructure is complete and the trajectory for any relocation task will be computed in time quadratic to the number of robots present in the system. In the following, we will assume that $G = (V, L)$ is a roadmap for a well-formed infrastructure $(\mathcal{W}, E)$.

Theorem 33. *If $G = (V, L)$ is a roadmap for a well-formed infrastructure $(\mathcal{W}, E)$ and COBRA with the time-extended roadmap planner is used to coordinate relocation tasks between the endpoints of the infrastructure, then all relocation tasks will be successfully carried out without collision.*

Proof. The line of argumentation used to prove the completeness of COBRA with an arbitrary complete trajectory planner (Theorem 32) is also applicable for COBRA with TERP – we only need to show that

Proposition 30 holds when TERP is used to find the best-response trajectory. The original argument can be adapted as follows: If the robot acquires an E-terminal token, there is a time point $\bar{t}$, when all robots in Φ reach their destinations and stay there. A best response trajectory for a relocation task $\mathbf{s} \to \mathbf{g}$ can always be constructed by waiting at the robot's start endpoint until $\left\lceil \frac{\bar{t}}{\delta} \right\rceil \delta$ (the first discrete time point, when all robots from Φ reach their terminal endpoint) and then by following the shortest path from $\mathbf{s}$ to $\mathbf{g}$ on roadmap G. □

Complexity

In this section, we derive the computational complexity of the COBRA algorithm when the TERP is used for finding the best-response trajectory. We will focus on the complexity of trajectory planning for a single relocation task and show that if a robot acquires a token with trajectories of other robots, all the robots in the token will reach their respective destination endpoints in a relative time that is bounded by a factor linearly dependent on the number of robots. Therefore, the size of the state space that needs to be searched during the trajectory planning is linear in the number of robots.

Assumptions and Notation: Let $f(\pi)$ denote the time when trajectory π reaches its destination, i.e., $f(\pi) := \min\, t \,\text{s.t.}\, t' \geq t :\, \pi(t') = \mathbf{g},\, \mathbf{g} \in E$. Further, let $A(\Phi, t)$ denote the set of "active" trajectories from token Φ at time t defined as $A(\Phi, t) := \{\pi : (\cdot, \pi) \in \Phi \,\text{s.t.}\, f(\pi) \geq t\}$. The latest time when all trajectories in token Φ reach their destination endpoints is given by function $F(\Phi) := \max_{(\cdot,\pi)\in\Phi} f(\pi)$.

Let v denote the number of vertices in the spatial roadmap. Further, let r denote the duration of the longest trajectory a robot can take between arbitrary two endpoints in the absence of dynamic obstacles (other robots). Observe that such a trajectory will consist of at most $|V|$ edges since otherwise the trajectory would visit one of the vertices twice, which contradicts its optimality. Let d denote the length of the longest edge in the spatial roadmap, $v_{\max}$ be the maximum speed of the robots,δ be the length of a timestep used in TERP, and k be the number of timesteps the longest edge in the time-extended roadmap spans computed as $k := \left\lceil \frac{d}{v_{\max}\cdot\delta} \right\rceil$. Then, the bound on the longest individually optimal trajectory can be obtained as $r = k\delta v$.

Lemma 34. *At any time point t we have $F(A(\Phi, t)) \leq t + |A(\Phi, t)|\, r$.*

Proof. Observe that the token only changes during new task handling. Let us consider an arbitrary sequence of tasks being handled by different robots at time points $\tau_1, \tau_2, \ldots$. Initially, at time $t = 0$ the token is empty and the inequality holds trivially: $F(\emptyset) \leq 0$. We will inductively show that the inequality holds after the token update during each such task handling, which implies that it also holds for the time period until the next update of the token. Now, let us take k^{th}-task handling by robot i with current trajectory π_i, that at time τ^k obtains token Φ^{k-1}. By induction hypothesis, we have $F(A(\Phi^{k-1}, \tau^k)) \leq \tau^k + \left|A(\Phi^{k-1}, \tau^k)\right| r$. We know that $\pi_i \notin A(\Phi^{k-1}, \tau^k)$, because the robot can only accept new tasks after it has finished the previous one and thus $f(\pi_i) < \tau^k$. Further, we know that 1) $\forall \pi \in \Phi^{k-1} \backslash A(\Phi^{k-1}, \tau^k)\ \forall t > \tau^k : \pi(t) \in E$ and 2) $\forall \pi \in A(\Phi^{k-1}, \tau^k)\ \forall t > F(A(\Phi^{k-1}, \tau^k)) \pi(t) \in E$, i.e. "inactive" trajectories are on the endpoints immediately, while active trajectories will reach their endpoints and stay there after $F(A(\Phi^{k-1}, \tau^k)$. Then, the robot finds its new trajectory $\pi_i^\star$, which can be in the worst-case constructed by waiting at the current endpoint and then following the shortest endpoint-avoiding path (which is in infrastructures guaranteed to exist and its duration is bounded by r) to the destination endpoint. Such a path reaches the destination in $\tau^k \leq f(\pi_i^\star) \leq F(A(\Phi^{k-1}, \tau^k) + r$. Then, the robot updates token $\Phi^k \leftarrow \Phi^{k-1} \setminus \{\pi_i\} \cup \{\pi_i^\star\}$. We know that $\pi_i \notin A(\Phi^{k-1}, \tau^k)$, but

$\pi_i^\star \in A(\Phi^k, \tau^k)$ and thus $\left|A(\Phi^k, \tau^k)\right| = \left|A(\Phi^{k-1}, \tau^k)\right| + 1$. By rearrangement

$$\begin{aligned} F(A(\Phi^{k-1}, \tau^k)) &\leq \tau^k + \left|A(\Phi^{k-1}, \tau^k)\right| r \\ F(A(\Phi^k, \tau^k)) &\leq F(A(\Phi^{k-1}, \tau^k)) + r \\ F(A(\Phi^k, \tau^k)) &\leq \tau^k + \left|A(\Phi^{k-1}, \tau^k)\right| r + r. \\ F(A(\Phi^k, \tau^k)) &\leq \tau^k + \left|A(\Phi^{k-1}, \tau^k) + 1\right| r \\ F(A(\Phi^k, \tau^k)) &\leq \tau^k + \left|A(\Phi^k, \tau^k)\right| r \end{aligned}$$

□

Theorem 35. *The worst-case asymptotic complexity of a single relocation task handling using COBRA with time-extended roadmap planning is $O(n^2k^2v^4)$, where n is the number of robots in the system, v is the number of vertices in the roadmap graph, k is the maximum number of timesteps a single edge in time-extended roadmap spans computed as $k := \left\lceil \frac{d}{v_{\max} \cdot \delta} \right\rceil$ such that d is the length if longest edge in the spatial roadmap, $v_{\max}$ is the maximum speed, and δ is the time-discretization step.*

Proof. It is known that $F(A(\Phi, t)) \leq t + |A(\Phi, t)|\, r$. Token Φ is updated in such a way that it contains at most one record for each robot. Assume that robot i handles a new relocation task. Before planning, robot i removes its trajectory from token Φ and thus we have $|\Phi| \leq n - 1$. Since $A(\Phi, t) \subseteq \Phi$, we have $|A(\Phi, t)| \leq n - 1$ and using Lemma 34, we get $F(A(\Phi, t)) \leq t + (n-1)r$, i.e. all other robots will be at their respective destination endpoint at latest time $t + (n-1)r$. Subsequently, the dynamic obstacles corresponding to the robots in token Φ stop changing at time $t' \leq t + (n-1)r$. From Lemma 16 under substitution $m = (n-1)$ and using

$$\begin{aligned} \tau &= \max(0, t + (n-1)r - (t - t_{\text{planning}}))/\delta + kv \\ &= \max(0, (n-1)r - t_{\text{planning}})/\delta + kv \\ &\leq ((n-1)r)/\delta + kv \\ &\leq (n-1)kv + kv \\ &\leq nkv, \end{aligned}$$

we conclude that the trajectory will be found in time $O(n^2k^2v^3 + n^2k^2v^4) = O(n^2k^2v^4)$. □

6.3 Experimental Evaluation

In this section, we compare the performance of COBRA and reactive collision avoidance algorithm ORCA, which serves as a baseline due to its position as one of the most popular decentralized approaches for collision avoidance in large multi-robot teams with incrementally assigned tasks.

6.3.1 Environments

The two algorithms are compared in well-formed infrastructures constructed in three real-world environments Hall, Office, and Warehouse shown in Figure 6.3.1. The detailed description of the environments can be found in Section 4.6.1.

6.3.2 Experiment Setup

The execution of a multi-robot system is simulated using a multi-robot simulator. During each simulation, the given number of robots is created and each of them is successively assigned 4 relocation tasks to a randomly chosen unassigned endpoint. When the simulation is started, all the robots are initialized

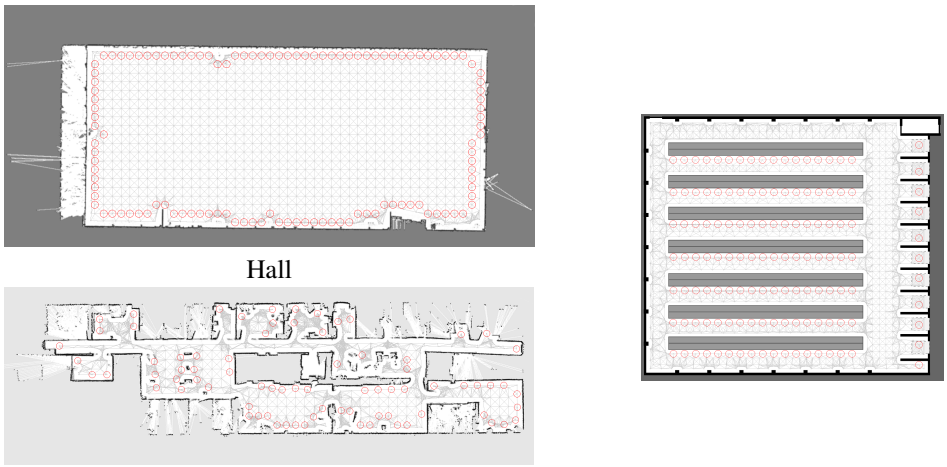


Figure 6.3.1: Test environments. The infrastructure endpoints depicted in red, the roadmap graph shown in gray.

and the first relocation task with random destination endpoint is issued. To avoid the initial unnatural situation in which all robots would need to plan simultaneously, the initial task is issued with a random delay within the interval [0, 30 s]. Once a robot reaches the destination of its task, a new random destination is generated and the process is repeated until the required number of relocation tasks have been generated. For each such simulation, we observe whether all robots successfully carried out all assigned tasks and the time needed to reach the destination of each relocation task. Further, we compute the prolongation introduced by collision avoidance as $p = t^A - t'$, where t^A is the duration of a particular task when an algorithm A is used for trajectory coordination, and t' is the time the robot needs to reach the destination without collision avoidance simply by following the shortest path at the roadmap at maximum speed.

The robots are modeled as circular bodies with radius r=50 cm that can travel at maximum speed $v_{\max}$=1 m/s. The relative size of a robot and the environment in which the robots operate is depicted in Figure 6.3.1, where the red circles indicate the size of a robot. The roadmaps are constructed as an 8-connected grid with additional vertices and edges added near the walls to maintain connectivity in narrow passages. The non-diagonal edges of the base grid are 130 cm long, the diagonal edges are 183 cm long.

The planning window used by COBRA is $t_{\text{planning}} = 3$ s long, yet on average single planning required less than 1 s even on the most challenging instances. The time-extended roadmap uses discretization δ=650 ms that conveniently splits travel on the non–diagonal edges into 2 timesteps and diagonal edges into 3 timesteps.

The reactive technique ORCA [van den Berg et al., 2011] is a control-engineering approach typically used in a closed-loop such that at each time instant it selects a collision-avoiding velocity vector from the continuous space of robot's velocities that is the closest to the robot's desired velocity. In our implementation, at each time instant, the algorithm computes a shortest path from the robot's current position to its goal on the same roadmap graph as is used for trajectory planning by COBRA. The desired velocity vector then points at this shortest path at the maximum speed. When using ORCA, we often witnessed dead-lock situations during which the robots either moved at extremely slow velocities or even stopped completely. If any of the robots did not reach its destination in the runtime limit of 10 mins (avg. task duration is less than 33 s), we considered the run as failed.

The source code of the algorithms, the simulator, and the test instances are available for download at

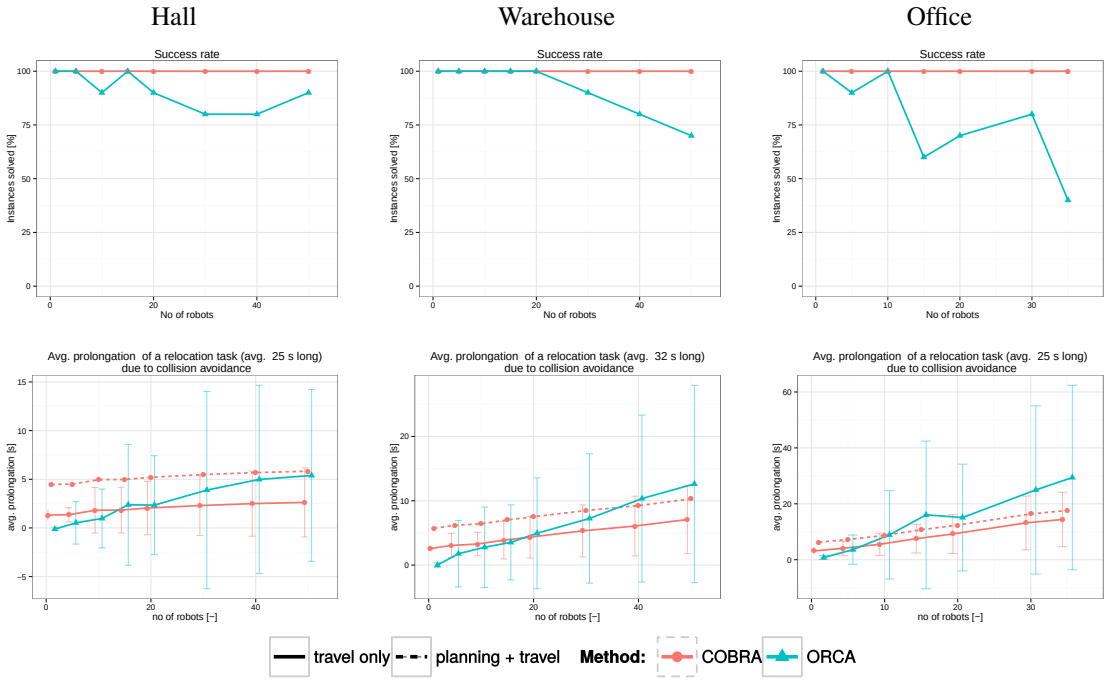


Figure 6.3.2: Results. The bars represent standard deviation.

`http://github.com/mcapino/cobra-icaps2015`. A video illustrating the behavior of the tested algorithms on selected instances is available at `https://www.youtube.com/watch?v=KUSPCOmX1ig`.

6.3.3 Results

Figure 6.3.2 shows the performance of COBRA and ORCA in the three test environments. The top row of plots shows the success rate of each algorithm on instances with increasing number of robots. In accordance with our theoretical results, we can see that the COBRA algorithm reliably leads all robots to their assigned destinations without collisions. When we tried to realize collision-free operation using ORCA, the algorithm led in some cases the robots into a dead-lock. The problem exhibited itself more often in environments with narrow passages as we can see in the success rate plot for the Office environment.

The bottom row of plots shows the average prolongation of a relocation task when either COBRA and ORCA is used for collision avoidance. The total prolongation introduced by COBRA is composed of two components: planning time and travel time. When a new task is issued, the robot waits for $t_{\text{planning}} = 3$ s in order to plan a collision-free trajectory to the destination of its new task. Only then, the found trajectory is traveled by the robot until the desired destination is reached. The robots controlled by ORCA start moving immediately because they follow a precomputed policy towards the destination of the current task or a collision-avoiding velocity if a possible collision is detected. Recall that ORCA performs optimization in the continuous space of robot's instantaneous velocities, whereas COBRA plans a global trajectory on a roadmap graph with a discretized time dimension. In the case of simple conflicts, ORCA can take advantage of its ability to optimize in the continuous space and generates motions where the robots closely pass each other, whereas COBRA has to stick to the underlying discretization, which does not always allow such close evasions. However, when the conflicts become more intricate, the advantages of global planning starts to outweigh the potential loss intro-

duced by space-time discretization. The exact influence of planning in a discretized space-time can be best observed by looking at the data point for instances with 1 robot – these instances do not involve any collision avoidance and thus the prolongation can be attributed purely to the discretization of space-time in which the robot plans. Further, we can observe that the local collision avoidance is less predictable, which is exhibited by the larger variation.

In this chapter, we have introduced a method for online coordination in a multi-robot system. The proposed algorithm, called Continuous Best-response Approach (COBRA), is specifically suited for coordination in well-formed infrastructures, where it is guaranteed to provide a collision-free trajectory to any relocation task in at most quadratic time in the number of robots. Our experiments have shown that the proposed method is more reliable and more efficient than the application of reactive methods for collision avoidance. In the next chapter, we will study the applicability of penalty methods for multi-robot motion planning and propose a penalty-based algorithm that can be used as a heuristic for improving the quality of the solution and the coverage of prioritized planning.

Chapter 7

Penalty Method for finding Near-optimal Solutions

One of the disadvantages of prioritized approaches is that the resulting trajectories can be noticeably suboptimal. In this chapter, we explore the applicability of the family of penalty-based approaches, which are commonly used in continuous mathematical optimization, to obtain high-quality solutions to the problem of batch trajectory coordination in multi-robot systems (Problem 6). In particular, we study the applicability of a penalty-based distributed optimization method proposed by Bhattacharya et al. [2010] that is based on the idea of increasing the penalty weights in so-called separable optimal flow direction at each iteration to obtain guarantees on convergence to a global optimum. The global optimality guarantees of the method rely on the assumption of *the existence of separable optimal flow direction at each step* that, as we find, is in general not satisfied in multi-robot trajectory planning problems. Based on these findings, we propose a simpler penalty-based method called k-step penalty method (kPM) and show that despite the lack of theoretical guarantees, the method can be used to find solutions that are of superior quality compared to trajectories generated by prioritized planning or reactive collision-avoidance approaches.

In the sequel, we shortly review some of the optimization approaches applied in the context of single and multi-robot trajectory planning. Then, in Section 7.2, we investigate the issue of the existence of separable optimal flow directions in multi-robot trajectory coordination problems. Finally, in Section 7.3 we propose a simple k-step penalty-based method for trajectory optimization and evaluate its performance against other commonly used methods for multi-robot trajectory coordination.

7.1 Optimization Approaches

An important class of methods for finding collision-free multi-robot trajectories is based on formulating the problem in the framework of continuous optimization. In particular, Schouwenaars et al. [2001] shown that a multi-robot path planning problem can be formulated and solved as a mixed-integer linear program (MILP) if a specific weighted one-norm cost function is used. More recently, mixed-integer quadratic programs (MIQP) were used to find collision-free trajectories for a team of quadcopters [Mellinger et al., 2012]. However, MILPs and MIQPs are costly to solve, which limits their applicability only to small multi-robot path planning problems. Chen et al. [2015] formulate and solve the problem using sequential convex programming techniques, which offers better scalability compared to MILP/MIQP-based approaches, but finds trajectories that are only locally optimal. Bhattacharya et al. [2010] propose a penalty-based optimization algorithm and apply it to obtain solutions to multi-robot rendezvous pathfinding problems. The algorithm iteratively replans the trajectory of each robot in a round-robin fashion such that at each iteration a penalty for violating the pairwise constraint on the tra-

jectories of two robots is slightly increased. As the penalties tend to infinity, the individual robots settle for constraint-abiding trajectories. It is shown that if the penalties are increased in a so-called *separable optimal flow direction* at each iteration, the algorithm will converge to a globally optimal solution. In the following section, we show that even though the algorithm was demonstrated to successfully work in a multi-robot path planning context, this was due to an incorrect implementation of the algorithm and these required directions for multi-robot trajectory planning problems do not exist in general.

7.2 On the Existence of Separable Optimal Flow

Bhattacharya et al. [2010] propose an elegant distributed optimization algorithm for multi-robot path planning problems with pairwise constraints and show that the algorithm is guaranteed to converge to a globally optimal solution if the weights associated with constraint violation penalties are increased in a so-called separable optimal flow direction. It is however not clear whether separable optimal flow directions exist in the multi-robot path planning setting. In this section, we show that these conditions in general do not hold for multi-robot path planning problems. In consequence, the algorithm, unfortunately, cannot be expected to provide globally optimal solutions, but rather ones that are close to being locally optimal.

In order to demonstrate why the global optimality guarantees are not applicable in the context of multi-robot path planning, we will have to reintroduce some of the notation from the original paper.

Problem definition

We start with the problem definition. Although in the original paper the optimization problem is stated in a general way, here we will pose the problem more specifically using the multi-robot path planning terminology.

Problem 36. The optimization problem we are going to discuss is

$$\{\pi_1^*, \ldots, \pi_n^*\} = \arg\min_{\pi_1,\ldots,\pi_n} \sum_{i=1,\ldots,n} c_i(\pi_i)$$

subject to the pairwise constraints

$$\Omega_{ij}(\pi_i, \pi_j) = 0, \quad \forall\, i, j = 1, \ldots, n,$$

where π_i is a trajectory of robot i, $c_i(\pi)$ is the cost robot i assigns to trajectory π, and $\Omega_{ij}(\pi_1, \pi_2)$ is a constraint function with a non-zero value if the constraint between trajectories π_1 and π_2 is violated. The problem assumptions are the following:

1. Optimization variables $\pi_1, \ldots, \pi_n$ represent trajectories that lie in a simply-connected space of continuous functions.
2. The cost functions $c_1, \ldots, c_n$ are continuous and smooth.
3. The constraint functions Ω_{ij} are defined for each unordered pair ij, $i, j = 1, \ldots N\, i \neq j$. They are assumed to have a form $\Omega_{ij}(\pi_j, \pi_i) = G_{ij}(\pi_i - \pi_j)$, where G_{ij} is continuous, smooth, and even function.

Notation

Now we repeat the notation used to manipulate the pairs of robots. Let $\mathcal{P}^n$ be the set of all unordered pairs of numbers $1, \ldots, n$, i.e.,

$$\mathcal{P}^n = \{\{1,2\}, \{1,3\}, \ldots, \{1,N\}, \{2,3\}, \{2,4\}, \ldots, \{N,r\}\}$$

and $\mathcal{P}_r^n$ be the subset of $\mathcal{P}^n$ such that one of the numbers of the unordered pair is r, i.e.

$$\mathcal{P}_r^n = \{\{1,r\},\ldots,\{r-1,r\},\{r+1,r\},\ldots,\{N,r\}\}.$$

The subscripts ij of quantities such as W or functions such as Ω are elements from $\mathcal{P}^N$ and thus the order does not matter. The weights for each pairwise constraint are represented as a vector W of length $\frac{1}{2}N(N-1)$ of all the W_{ij}'s. Also the expression $\{i,j\} \equiv \{k,r\} \in \mathcal{P}_r^n$ is used to indicate that $\{i,j\}$ is such that exactly one of i or j is equal to r, while the other, not equal to r, is denoted by k.

The Lagrangian, i.e., the penalized unconstrained version of Problem 36 is

$$\overline{U}(\{\pi\},W) := \sum_{k\in 1,\ldots,n} c_k(\pi_k) + \sum_{\{kl\}\in\mathcal{P}^n} W_{kl}\Omega_{kl}(\pi_k,\pi_l),$$

the optimal solution at penalty weights W is

$$\{\overline{\Pi}\}(W) := \arg\min_{\{\pi\}}[\overline{U}(\{\pi\},W)],$$

and the Dual function, i.e., the optimal value at penalty weights W, is

$$\overline{\Psi}(W) := \min_{\{\pi\}}[\overline{U}(\{\pi\},W)] = U(\{\overline{\Pi}\}(W),W).$$

Analogous notation system can be defined also for each robot i. We define

$$U_r(\pi_r,W_1,W_2) := c_r(\pi_r) + \sum_{\{kr\}\in\mathcal{P}_r^n} W_{1,kr}\Omega_{kr}(\overline{\Pi}_k(W_2),\pi_r).$$

Given the trajectories of other robots at penalty weights W_2, function $\Pi_r(W_1,W_2)$ gives the optimal trajectory for robot r at penalty weight W_1 and $\Psi_r(W_1,W_2)$ gives its cost:

$$\Pi_r(W_1,W_2) := \arg\min_{\pi}[U_r(\pi,W_1,W_2)]$$

$$\Psi_r(W_1,W_2) := \min_{\pi}[U_r(\pi,W_1,W_2)] = U(\Pi_r(W_1,W_2),W_1,W_2).$$

Distributed Optimization Algorithm

The proposed algorithm translates the original constrained problem (Problem 36) into an unconstrained problem, where the pairwise constraints are modeled by respective penalty terms in the objective function. The penalty associated with the violation of the constraint between the pair of robots i and j is controlled using a penalty weight parameter W_{ij}. At the beginning, all the penalty weights are set to zero and the algorithm finds an optimal trajectory for each robot considering no interactions between the robots. Then, the algorithm proceeds in iterations until all constraints are (almost fully) satisfied. In each iteration, one of the robots is selected, the penalty weights are slightly increased in a prescribed way, and a trajectory that optimizes the sum of cost and the increased penalty is computed for the selected robot, while the trajectories of the other robots remain fixed. In the subsequent iteration, another robot is selected, weight-increase direction is chosen according to a prescribed method and a new trajectory for the chosen robot is computed. With the increasing number of iterations, the penalty weights tend to infinity, subsequently the penalty term starts to dominate the cost term, and the robots' optimal trajectories have lower levels of constraint violation converging to a feasible solution. The pseudocode of the algorithm is in Figure 10.

On of the central considerations in the paper presenting the algorithm is how to increase the weights throughout the iterative process. At each iteration of the algorithm, the weights are increased with a specific step size ϵ^k and step direction V^k. The step size is for the purposes of theoretical analysis assumed

Algorithm 10: Distributed Optimization Algorithm

Algorithm `DOA`
- $\pi_i^0 \leftarrow \arg\min_\pi c_i(\pi) \quad \forall i = 1 \dots n$;
- set $W_{ij}^0 \leftarrow 0$ for all unordered pairs $\{i, j\} \in \mathcal{P}^n$;
- set $r = r_0$ to an arbitrary robot $1, \dots, n$
- $k \leftarrow 0$
- **while** $\Omega_{ij}(\pi_i^k, \pi_j^k) \neq 0 \quad \forall \{i, j\} \in \mathcal{P}^n$ **do**
  - $V^k \leftarrow$ `ComputeStepDirection(`$W^k, \{\pi\}^k, r$`)`;
  - $W^{k+1} \leftarrow W^k + \epsilon^k V^k$;
  - $\pi_r^{k+1} \leftarrow \arg\min_\pi [c_r(\pi) + \Sigma_{\{ir\} \in \mathcal{P}_r^n} W_{ir}^{k+1} \Omega_{ir}(\pi_i^k, \pi)]$;
  - $\pi_j^{k+1} \leftarrow \pi_j^k \quad \forall j \neq r$;
  - $k \leftarrow k + 1$;
  - select $r = r_{k+1} \in \{1, \dots n\}$ such that all robots get processed approximately equally often

to be infinitesimal, in practice the step size can be a small fixed constant or chosen adaptively. The step direction V^k is computed using `ComputeStepDirection()` function and the choice of the direction is a crucial ingredient of the presented algorithm. In an involved theoretical analysis, the authors show that if the V^k and ϵ^k is chosen such that they constitute a so-called *Ascent Separable Optimal Flow Direction* at W_k for robot r_k at each iteration of the algorithm k, then the algorithm is guaranteed to converge to a globally optimal solution if it exists. Moreover, the authors give a method that allows computing such a direction if it exists using quantities that are often readily available, namely: W^k, $c_i^{(2)} \quad \forall i \in 1, \dots, n$, and $\Omega_{ij}^{(0,2)}(\pi_i^k, \pi_j^k), \Omega_{ij}^{(1,1)}(\pi_i^k, \pi_j^k), \Omega_{ij}^{(1,0)}(\pi_i^k, \pi_j^k) \quad \forall \{i, j\} \in \mathcal{P}^n$. It is however not known whether such directions indeed exist and whether their existence can be guaranteed for some class of multi-robot path planning problems.

Separable Optimal Flow

The global optimality of the discussed technique relies on the existence of separable optimal flow direction at each step. The notion of separable optimal flow is defined [Bhattacharya et al., 2010, Definition 1] as

Definition 37. Given the functions c_r, $\Omega_{ir} \quad \forall \{ir\} \in \mathcal{P}_r^n$, we call V a Separable Optimal Flow Direction and ϵ a Separable Optimal Flow Step at W for Ψ_r if and only if

$$\Psi_r(W + \epsilon V, W) - \Psi_r(W, W) \leq \Psi_r(W + \epsilon V, W + \epsilon V) - \Psi_r(W, W + \epsilon V)$$

and $V_{ij} = 0 \quad \forall \{ij\}$ such that $r \notin \{i, j\}$. Together, V and ϵ are said to define a Separable Optimal Flow at W for Ψ_r.

In the proof of Theorem 3 in [Bhattacharya et al., 2010], it is shown that Separable Optimal Flow Directions are linear combinations of the right eigenvectors corresponding to the non-negative eigenvalues of the matrix $\Psi_r^{(1,1)}(W, W)$. Therefore, Separable Optimal Flow at W for Ψ_r exists precisely when $\Psi_r^{(1,1)}(W, W)$ has positive eigenvalues. Although $\Psi_r^{(1,1)}(W, W)$ can be computed numerically using finite differences, such a process would involve evaluating $\{\overline{\Pi}\}(W)$ for a number of different values of W, which corresponds to solving the original unconstrained problem multiple times. Therefore, in the proof of Theorem 3, a method for computing $\Psi_r^{(1,1)}(W, W)$ from quantities that are readily available is devised.

The applicability of the method is demonstrated [Bhattacharya et al., 2010, Section 5-A] using a 1-dimensional N-robot rendezvous problem. In this problem, the trajectory of each robot is defined as

a vector having length T representing the displacement of each robot at time points $1, \dots N$ under the constraint that given pairs of robots need to meet at specified time points. The algorithm is reported to approach the optimal solution for this problem. Indeed, when the provided Matlab implementation of the algorithm is executed, it can be observed that the algorithm is able to find a weight increase direction that is allegedly a separable optimal flow direction at each step of the iterative process.

2-point Rendezvous Problem

The above example problem is unfortunately still too complex to allow analytic computation of $\Psi_r(W_1, W_2)$ and its derivatives to cross-check against the results obtained using the proposed indirect method with an analytic solution. To allow direct analytic computation of the $\Psi_r(W_1, W_2)$ and to assist intuition, we will focus on probably the simplest possible version of multi-robot rendezvous problem. The considered problem involves two robots constrained to move on a line and the trajectory of each robot has only one time step. Such a simplistic problem can be formulated as follows.

Problem 38. Consider 2 points moving on a 1-d line. The position of point i is denoted as $\pi_i \in \mathbb{R}$. The desired position of robot 1 is $\pi_1 = -1$ and the desired position of robot 2 is $\pi_2 = 1$. The task is to place the points on the line such that the sum of the squared distances of each point from their desired position is minimized and their position is equal, i.e. the robots rendezvous:

$$\begin{aligned} &\underset{\pi_1,\pi_2\in\mathbb{R}}{\text{minimize}} && (\pi_1 - (-1))^2 + (\pi_2 - 1)^2 && \text{(distance from initial position)} \\ &\text{subject to} && \pi_1 = \pi_2 && \text{(rendezvous)} \end{aligned},$$

When we replace the rendezvous constraint with a penalty term $\Omega_{12}(\pi_1, \pi_2) = (\pi_1 - \pi_2)^2$, then we get the Lagrangian

$$\overline{U}(\{\pi_1, \pi_2\}, w) := (\pi_1 + 1)^2 + (\pi_2 - 1)^2 + w \cdot (\pi_1 - \pi_2)^2$$

with the optimal solution obtained as

$$\overline{\{\pi_1, \pi_2\}} = \underset{\{\pi_1,\pi_2\}}{\arg\min}\, \overline{U}(\{\pi_1, \pi_2\}, w)$$

and the dual function

$$\overline{\Psi}(w) = \min_{\{\pi_1,\pi_2\}} \overline{U}(\{\pi_1, \pi_2\}, w).$$

Computing Required Derivatives

Thanks to the simple formulation of the problem all the required derivatives can be obtained analytically:

$$\begin{aligned} c_1(\pi) &= (\pi + 1)^2 \\ c_1^{(1)}(\pi) &= 2(\pi + 1) \\ c_1^{(2)}(\pi) &= 2 \end{aligned}$$

$$\begin{aligned} c_2(\pi) &= (\pi - 1)^2 \\ c_2^{(1)}(\pi) &= 2(\pi - 1) \\ c_2^{(2)}(\pi) &= 2 \end{aligned}$$

$$\Omega_{12}(\pi_1,\pi_2) = \Omega_{21}(\pi_1,\pi_2) = (\pi_1-\pi_2)^2$$
$$\Omega_{12}^{(1,0)}(\pi_1,\pi_2) = \Omega_{21}^{(1,0)}(\pi_1,\pi_2) = 2(\pi_1-\pi_2)\cdot 1 = 2(\pi_1-\pi_2) = 2\pi_1 - 2\pi_2$$
$$\Omega_{12}^{(0,1)}(\pi_1,\pi_2) = 2(\pi_1-\pi_2)\cdot(-1) = -2(\pi_1-\pi_2) = -2\pi_1 + 2\pi_2$$
$$\Omega_{12}^{(2,0)}(\pi_1,\pi_2) = 2$$
$$\Omega_{12}^{(0,2)}(\pi_1,\pi_2) = 2$$
$$\Omega_{12}^{(1,1)}(\pi_1,\pi_2) = -2$$

Computing Optimal Solution

The optimal solution for the Lagrangian of the 2-point rendezvous problem can be computed analytically as well. First, we compute the gradient for a fixed weight w as

$$\frac{\partial}{\partial\pi_1}\overline{U}((\{\pi_1,\pi_2\},w) = (2+2w)\pi_1 - 2w\pi_2 + 2$$
$$\frac{\partial}{\partial\pi_2}\overline{U}((\{\pi_1,\pi_2\},w) = (2+2w)\pi_2 - 2w\pi_1 - 2.$$

Then, we find π_1 (π_2) where the gradient of the Lagrangian is zero

$$\frac{\partial}{\partial\pi_1}\overline{U}((\{\pi_1,\pi_2\},w) = 0 \text{ for } \pi_1 = \frac{w\pi_2 - 1}{1+w}$$
$$\frac{\partial}{\partial\pi_2}\overline{U}((\{\pi_1,\pi_2\},w) = 0 \text{ for } \pi_2 = \frac{w\pi_1 + 1}{1+w}$$

and solve the corresponding linear system

$$\begin{array}{cccc} (1+w)\pi_1 & -w\pi_2 & 2 & = 0 \\ -w\pi_1 & (1+w)\pi_2 & -2 & = 0 \end{array},$$

which yields the optimal solution for the unconstrained version of 2-point rendezvous problem in the analytic form

$$\overline{\Pi}_1(w) = \frac{-1}{1+2w}$$
$$\overline{\Pi}_2(w) = \frac{1}{1+2w}.$$

Computing $\Psi_1^{(1,1)}(w,w)$

The mixed derivative of $\Psi(w,w)$ can be also computed analytically. The solution $\Pi_1(w_1,w_2) := \arg\min_{\pi} U(\pi,w_1,w_2)$ can be computed by differentiating

$$\frac{\partial}{\partial\pi}U_1(\pi,w_1,w_2) = 2\pi + w_1 2\pi + 2 - \frac{2w_1}{1+2w_2}$$

and finding a point π, where the derivative is zero

$$\frac{\partial}{\partial\pi}U_1(\pi,w_1,w_2) = 0 \text{ for } \pi = \frac{\frac{2w_1}{1+2w_2} - 2}{2+w_1},$$

which yields

$$\Pi_1(w_1, w_2) = \frac{\frac{2w_1}{1+2w_2} - 2}{2 + w_1}.$$

The function $\Psi_1(w_1, w_2)$ can be expressed as

$$\Psi_1(w_1, w_2) = \left(\frac{\frac{2w_1}{1+2w_2} - 2}{2 + w_1} + 1 \right)^2 + w_1 \left(\frac{\frac{2w_1}{1+2w_2} - 2}{2 + w_1} - \frac{1}{1 + 2w_2} \right)^2,$$

its derivation with respect to w_1 as

$$\Psi_1^{(1,0)}(w_1, w_2) = \frac{6w_1^2 + w_1^3 + 32(1 + w_2)^2 - 4w_1(3 + 4w_2)}{(2 + w_1)^3(1 + 2w_2)^2},$$

and finally the mixed derivative as

$$\Psi_1^{(1,1)}(w_1, w_2) = \frac{-4(6w_1^2 + w_1^3 + 16(1 + w_2) - 8w_1(1 + w_2))}{(2 + w_1)^3(1 + 2w_2)^3}.$$

When we set $w_1 = w_2 = w$ then we get

$$\Psi_1^{(1,1)}(w, w) = \frac{-4(w^3 - 2w^2 + 8w + 16)}{(2 + w)^3(1 + 2w)^3}. \tag{7.2.1}$$

The separable optimal flow exists only if $\Psi_1^{(1,1)}(w, w)$ has a positive eigenvalue. Since in our case, the mixed derivative $\Psi_1^{(1,1)}(w, w)$ is a scalar, the separable optimal flow direction for robot 1 for a particular weight w may exist only if $\Psi_1^{(1,1)}(w, w)$ is positive. By solving the equation

$$\frac{-4(w^3 - 2w^2 + 8w + 16)}{(2 + w)^3(1 + 2w)^3} \geq 0,$$

we find out that the above inequality cannot be satisfied for $w > 0$. Consequently, for any positive weight w, we have $\Psi_1^{(1,1)}(w, w) < 0$ and thus even for our simple 2-point rendezvous problem there are no separable optimal flow directions for robot 1.

We implemented the discussed algorithm (Algorithm 10) such that `ComputeStepDirection()` follows the method described in the proof of Theorem 3 and applied it to our 2-point rendezvous problem. To compute $\Psi_1^{(1,1)}(w, w)$ from quantities w, $c_1^{(2)}$, $c_2^{(2)}$,$\Omega_{12}^{(0,2)}(\pi_1, \pi_2)$, $\Omega_{12}^{(1,1)}(\pi_1, \pi_2)$, and $\Omega_{12}^{(1,0)}(\pi_1, \pi_2)$ we proceeded in accordance with the proposed method [Bhattacharya et al., 2010, proof of Theorem 3] as follows.

First, we compute the values of $M_1(w)$, $M_2(w)$ and $N_{12}(w)$ as

$$\begin{aligned} M_1(w) &= c_1^{(2)}(\overline{\Pi}_1(w)) + w\Omega_{12}^{(0,2)}(\overline{\Pi}_2(w), \overline{\Pi}_1(w)) \\ &= 2 + 2w, \end{aligned}$$

$$\begin{aligned} M_2(w) &= c_2^{(2)}(\overline{\Pi}_2(w)) + w\Omega_{12}^{(0,2)}(\overline{\Pi}_1(w), \overline{\Pi}_2(w)) \\ &= 2 + 2w, \end{aligned}$$

$$\begin{aligned} N_{12}(w) &= \Omega_{12}^{(1,1)}(\overline{\Pi}_1(w), \overline{\Pi}_2(w)) \\ &= -2. \end{aligned}$$

Then values of $\overline{\Pi}_1^{(1)}(w)$ and $\overline{\Pi}_2^{(2)}(w)$ can be obtained as

$$\begin{bmatrix} \overline{\Pi}_1^{(1)}(w) \\ \overline{\Pi}_2^{(1)}(w) \end{bmatrix} = \begin{bmatrix} M_1(w) & wN_{12}(w) \\ wN_{21}(w) & M_2(w) \end{bmatrix}^{-1} \cdot \begin{bmatrix} \Omega_{12}^{(1,0)}(\overline{\Pi}_1(w), \overline{\Pi}_2(w)) \\ \Omega_{12}^{(1,0)}(\overline{\Pi}_2(w), \overline{\Pi}_1(w)) \end{bmatrix}$$
$$\begin{bmatrix} \overline{\Pi}_1^{(1)}(w) \\ \overline{\Pi}_2^{(1)}(w) \end{bmatrix} = \begin{bmatrix} 2+2w & -2w \\ -2w & 2+2w \end{bmatrix}^{-1} \cdot \begin{bmatrix} 2(\overline{\Pi}_1(w) - \overline{\Pi}_2(w)) \\ 2(\overline{\Pi}_2(w) - \overline{\Pi}_1(w)) \end{bmatrix}. \tag{7.2.2}$$

The values of $\Pi_1^{(1,0)}(w,w)$ and $\Pi_2^{(1,0)}(w,w)$ can be obtained as

$$\begin{aligned} [\Pi_1^{(1,0)}(w,w)] &= [M_1(w)]^{-1} \cdot \Omega_{21}^{(1,0)}(\overline{\Pi}_2(w), \overline{\Pi}_1(w)) \\ &= [2+2w]^{-1}(2\overline{\Pi}_2(w) - 2\overline{\Pi}_1(w)) \\ &= [2+2w]^{-1}2(\overline{\Pi}_2(w) - \overline{\Pi}_1(w)) \end{aligned}$$

$$\begin{aligned} [\Pi_2^{(1,0)}(w,w)] &= [M_2(w)]^{-1} \cdot \Omega_{12}^{(1,0)}(\overline{\Pi}_1(w), \overline{\Pi}_2(w)) \\ &= [2+2w]^{-1}2(\overline{\Pi}_2(w) - \overline{\Pi}_1(w)). \end{aligned}$$

And finally $\Psi_1^{(1,1)}(w,w)$ is computed as

$$\begin{aligned} \Psi_1^{(1,1)}(w,w) =& \Omega_{21}^{(1,0)}(\overline{\Pi}_2(w), \overline{\Pi}_1(w)) \cdot (\overline{\Pi}_2^{(1)}(w) - \overline{\Pi}_1^{(1)}(w)) + \Omega_{21}^{(1,0)}(\overline{\Pi}_2(w), \overline{\Pi}_1(w)) \cdot (\Pi_r^{(1,0)}(w,w)) \\ =& -2(\overline{\Pi}_2(w) - \overline{\Pi}_1(w)) \cdot (\overline{\Pi}_2^{(1)}(w) - \overline{\Pi}_1^{(1)}(w)) - 2(\overline{\Pi}_2(w) - \overline{\Pi}_1(w)) \cdot (\Pi_1^{(1,0)}(w,w)). \end{aligned}$$

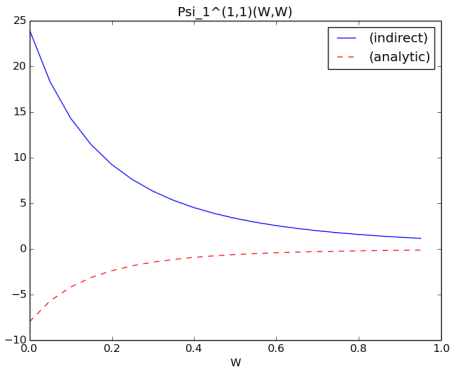
Surprisingly, we found out that the procedure does return a direction V for every weight w that is allegedly a separable optimal flow direction, which contradicts our observation that $\Psi_1^{(1,1)}(w,w)$ is negative for any positive weight and therefore separable optimal flow directions do not exists for robot 1. We investigated this inconsistency and found out that values of $\Psi_1^{(1,1)}(w,w)$ for a particular weight w when computed using the method described in the proof of Theorem 3 in [Bhattacharya et al., 2010] does not correspond to the value we obtained by evaluating its analytic form (Formula 7.2.1). The plot of function $\Psi_1^{(1,1)}(w,w)$ obtained 1) using the described indirect method and 2) by evaluating the analytic formula are shown in Figure 7.2.1a.

The cause for this inconsistency can be traced down to the step between formulas (26) and (27) in [Bhattacharya et al., 2010] that prescribe how to compute derivatives of $\overline{\Pi}_r(w)$ from $\overline{\Pi}_r$ and the derivatives of c_r and Ω_{ij}. During this step, the equation (26), which reads

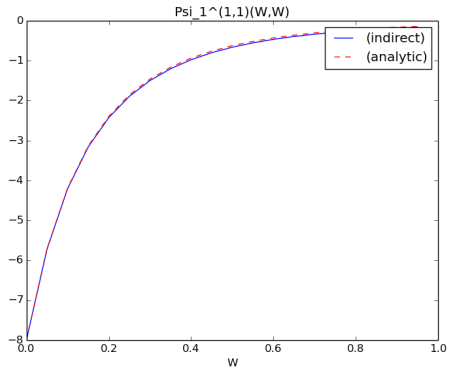
$$\begin{aligned} & M_r(W) \cdot \left[\overline{\Pi}_r^{(1)}(W)\right]_{ij} + \sum_{\{lr\} \in \mathcal{P}_r^N} W_{lr} N_{lr}(W) \cdot \left[\overline{\Pi}_l^{(1)}(W)\right]_{ij} \\ &= \begin{cases} \Omega_{kr}^{(1,0)}(\overline{\Pi}_k(W), \overline{\Pi}_r(W)) & \text{when } \{i,j\} \equiv \{k,r\} \in \mathcal{P}_r^N \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

is rearranged to the equation (27), i.e.,

$$\begin{bmatrix} M_1(W) & W_{12}N_{12}(W) & W_{13}N_{13}(W) & \cdots \\ W_{21}N_{21}(W) & M_2(W) & W_{23}N_{23}(W) & \cdots \\ W_{31}N_{31}(W) & W_{32}N_{32}(W) & M_3(W) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \left[\overline{\Pi}_1^{(1)}(W)\right]_{ij} \\ \left[\overline{\Pi}_2^{(1)}(W)\right]_{ij} \\ \left[\overline{\Pi}_3^{(1)}(W)\right]_{ij} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ 0 \\ \Omega_{mn}^{(1,0)}(\overline{\Pi}_m(W), \overline{\Pi}_n(W)) \\ 0 \\ \vdots \\ 0 \\ \Omega_{mn}^{(1,0)}(\overline{\Pi}_n(W), \overline{\Pi}_m(W)) \\ 0 \\ \vdots \end{bmatrix} \begin{matrix} \\ \\ \rightarrow m^{\text{th}} \text{ row} \\ \\ \\ \\ \rightarrow n^{\text{th}} \text{ row} \\ \\ \\ \end{matrix}$$



(a) The plot of $\Psi_1^{(1,1)}(w,w)$ obtained using a) the indirect method and b) analytically.



(b) The plot of $\Psi_1^{(1,1)}(w,w)$ obtained using a) the indirect method when equation 7.2.2 has been corrected and b) by evaluating the analytically form of the function.

Figure 7.2.1: Computing $\Psi_1^{(1,1)}(w,w)$

for all $\{m,n\} \in \mathcal{P}^N$. Careful verification of the rearrangement reveals that the m^{th} and n^{th}row of the vector on the right side of the equation (27) are mistakingly swapped. The valid rearrangement of the equation (26) is instead

$$\begin{bmatrix} M_1(W) & W_{12}N_{12}(W) & W_{13}N_{13}(W) & \cdots \\ W_{21}N_{21}(W) & M_2(W) & W_{23}N_{23}(W) & \cdots \\ W_{31}N_{31}(W) & W_{32}N_{32}(W) & M_3(W) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \left[\overline{\Pi}_1^{(1)}(W)\right]_{ij} \\ \left[\overline{\Pi}_2^{(1)}(W)\right]_{ij} \\ \left[\overline{\Pi}_3^{(1)}(W)\right]_{ij} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ 0 \\ \Omega_{mn}^{(1,0)}(\overline{\Pi}_n(W), \overline{\Pi}_m(W)) \\ 0 \\ \vdots \\ 0 \\ \Omega_{mn}^{(1,0)}(\overline{\Pi}_m(W), \overline{\Pi}_n(W)) \\ 0 \\ \vdots \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \rightarrow n^{\text{th}} \text{ row} \\ \\ \\ \end{matrix} .$$

We can check the correctness of this rearrangement in our simple 2-point rendezvous problem, by replacing the equation 7.2.2 that is used to compute $\Psi_1^{(1,1)}(w,w)$ with its corrected version

$$\begin{bmatrix} \overline{\Pi}_1^{(1)}(w) \\ \overline{\Pi}_2^{(1)}(w) \end{bmatrix} = \begin{bmatrix} M_1(w) & wN_{12}(w) \\ wN_{21}(w) & M_2(w) \end{bmatrix}^{-1} \cdot \begin{bmatrix} \Omega_{12}^{(1,0)}(\overline{\Pi}_2(w), \overline{\Pi}_1(w)) \\ \Omega_{12}^{(1,0)}(\overline{\Pi}_1(w), \overline{\Pi}_2(w)) \end{bmatrix}$$

$$\begin{bmatrix} \overline{\Pi}_1^{(1)}(w) \\ \overline{\Pi}_2^{(1)}(w) \end{bmatrix} = \begin{bmatrix} 2+2w & -2w \\ -2w & 2+2w \end{bmatrix}^{-1} \cdot \begin{bmatrix} 2(\overline{\Pi}_2(w) - \overline{\Pi}_1(w)) \\ 2(\overline{\Pi}_1(w) - \overline{\Pi}_2(w)) \end{bmatrix} .$$

When the value of $\Psi_1^{(1,1)}(w,w)$ is computed with the above correction, the values obtain using the indirect method are consistent with the values obtained by evaluating the analytic form of $\Psi_1^{(1,1)}(w,w)$ as is demonstrated in Figure 7.2.1b.

The consequences of the minor mistake in equation (27) in [Bhattacharya et al., 2010] are unfortunately rather major. As we could see already in our simple 2-point rendezvous scenario, the function $\Psi_1^{(1,1)}(w,w)$ is never positive for a positive weight, therefore there are no separable optimal flow directions for robot 1 and consequently, the proposed algorithm is unable to guarantee convergence to a globally optimal solution even for probably the simplest formulation of multi-robot rendezvous problem.

Moreover, when we fixed the above issue in the Matlab implementation of the multi-robot rendezvous problem as described in [Bhattacharya et al., 2010, Section V-A], we found that the corrected version of the algorithm implementation also fails to find separable optimal flow directions to follow in the presented example problem.

Finally, we attempted to confirm the existence of separable optimal on a simple formulation of a multi-robot trajectory coordination problem, where the robots must maintain minimum separation and found out that in such a problem too, separable optimal flow does not exist.

Our results suggest that there is no specific reason to believe that separable optimal flow directions should exist in multi-robot trajectory planning problems. Yet, it remains to be concluded whether such a property cannot hold for some fundamental reason or if there are some non-trivial multi-robot problems that allow the existence of such directions. We note that separable optimal flow can be trivially shown to exist, e.g. if all Ω_{ij} functions are zero, i.e. the robots' trajectories are independent with respect to the global cost.

7.3 k-step Penalty Method

Motivated by the above findings, in this section, we will study a simplified penalty-based algorithm for multi-robot motion planning called k-step Penalty Method (kPM) that instead of computing and following the separable optimal flow direction at each iteration, it increases the penalty weight uniformly by a small quantity on all pairwise constraints. The proposed algorithm can be also seen as a generalization of the prioritized planning framework that replaces hard constraints at each iteration by a soft constraint in an attempt to improve solution quality and instance coverage, while retaining computational tractability.

In the proposed approach, the requirement on the minimal separation between two trajectories is modeled by a penalty function that assigns a penalty to each pair of trajectories based on how much they violate the separation requirement. The solution to the multi-robot pathfinding problem is constructed by gradually increasing the penalty assigned to a robot when it passes through a collision region and by letting each robot replan its trajectory to account for the increased penalty.

Initially, the algorithm ignores interactions between the robots and finds an individually optimal trajectory for each robot. Then, the algorithm starts gradually increasing the penalty. After each increase in weight, one of the robots is selected and a new optimal trajectory that reflects the increased penalty is found for the robot. The process is repeated until the penalties are large enough to start dominating the cost of trajectories, effectively forcing them out of the collision regions.

The minimal separation constraints between a pair of robots i, j is approximated by a penalty function assigning a penalty to each part of the trajectory of robot i that gets closer to the trajectory of robot j than the required separation distance $d_{ij}^{sep} = r_i + r_j$. The penalty function has the form

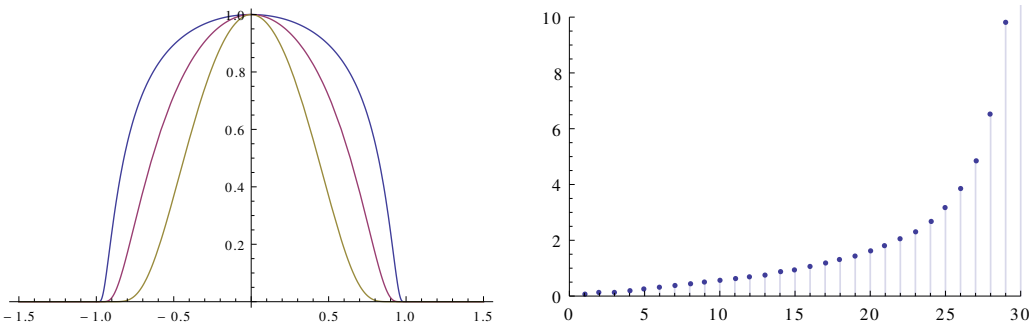
$$\Omega_{ij}(\pi_i, \pi_j) = \int_0^\infty \omega_{ij}(|\pi_i(t) - \pi_j(t)|)\, dt\,,$$

where $\omega_{ij}(d) : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a continuous function assigning a time-point penalty for interaction of the trajectories π_i and π_j at a time point t. In the case the trajectories do not violate the separation distance d_{sep} at the time point t, the function assigns zero penalty. Generally, the function needs to satisfy the following conditions:

$$\forall d \quad \begin{cases} \omega_{ij}(d) = 0 & \text{if } d \geq d_{ij}^{sep} \\ \omega_{ij}(d) > 0 & \text{if } d < d_{ij}^{sep} \end{cases}.$$

An example of a smooth function that satisfies these conditions is a bump function

$$\omega_{ij}(d) = \begin{cases} \frac{p_{max}}{e^{-s}} \cdot e^{-\frac{s}{1-(d/d_{ij}^{sep})^2})} & \text{for } d < d_{ij}^{sep} \\ 0 & \text{otherwise} \end{cases},$$



(a) Three bump functions with parameters $p_{max} = 1$, $d_{ij}^{sep} = 1$ and $s = 1/3$ (blue), $s = 1$ (purple) and $s = 3$ (yellow)

(b) Example 30 elements long weight sequence $w_1, \ldots, w_{30}$

Figure 7.3.1: Penalty function and weight sequence function

where p_{max} is a constant that can be used to adjust the penalty at $d = 0$ and s is a constant that can be used to adjust the "steepness" of the function. The intuition is that the "more" the vehicles violate the separation constraint in a given time-point, that is, the closer they get, the higher penalty should be assigned to the violation in that time-point. Figure 7.3.1a depicts example plots of three bump functions having different values of the parameter s.

Algorithm 11 exposes the *k-step Penalty Method* (kPM) algorithm that replans the trajectory of each robot exactly k-times. The algorithm starts by finding a cost-optimal trajectory for each robot using $w = 0$, i.e., while ignoring interactions with other robots. Then, it gradually increases the weight w and thus the penalties start to be taken into account. After each increase of the weight coefficient, one of the robots is selected and its trajectory is replanned to account for the increased penalty. After the iterative phase finishes, the trajectories of all robots are replanned for the last time with the weight coefficient set to infinity. If the final set of trajectories is conflict-free, the algorithm returns the trajectories as a valid solution, otherwise, it reports failure.

Since each robot replans once in the beginning (line 3) and once at the end (line 9) of the algorithm, the robot is left with $k - 2$ opportunities for replanning in the iterative phase of the algorithm. Hence, it performs $n(k - 2)$ iterations in the iterative stage with all the robots taking turns in a round-robin fashion (line 7).

During the iterative phase, the weights are increased in a sequence $w_1, \ldots, w_l$, with $l = n(k - 2)$. To model the gradual hardening of the separation constraints ultimately leading to the hard separation constraint, the sequence needs to converge to infinity. One way of generating such a sequence is using functions of the following form $w_i = \tan(\frac{i}{l+1} \cdot \frac{\pi}{2})$ for $i = 1, \ldots, l$. Figure 7.3.1b shows an example plot of such a sequence.

In each iteration of the algorithm, a selected robot can generally respond in two ways to a weight increase. On the one hand, it can accept the increase and simply leave its trajectory unchanged. On the other hand, in can find a higher-cost trajectory that avoids the penalized region. The optimal trajectory for the robot, however, often lies in between these two extremes and avoids the penalty region only partially such that the increased cost of the trajectory with decreased received penalty are optimally traded off. This corresponds to finding an optimal trajectory in a spatio-temporal cost field. To find such a trajectory we discretize the free space in form of a grid-like graph and add a discretized time dimension. An optimal discrete solution is then obtained by running the A* algorithm on such a space-time graph.

The penalty method has two benefits over prioritized planning. First, the penalty method can solve instances that are out of reach of prioritized planning. For example, the corridor swap scenario (introduced in Figure 2.2.2) can be resolved by the penalty method in $k = 10$ iterations. Figure 7.3.2 illus-

Algorithm 11: k-step Penalty Method

Algorithm `PM(`k`)`
  for $i \leftarrow 1 \dots n$ **do**
    $\pi_i \leftarrow$`Replan(`$i, 0$`)`;
  for $i \leftarrow 1 \dots n(k-2)$ **do**
    $r \leftarrow i \bmod n(k-2)$;
    $w_i \leftarrow \tan(\frac{i}{n(k-2)+1} \cdot \frac{\pi}{2})$;
    $\pi_i \leftarrow$`Replan(`i, w_i`)`;
  for $i \leftarrow 1 \dots n$ **do**
    $\pi_i \leftarrow$`Replan(`i, ∞`)`;
  if $\forall_{ij}\ \Omega_{ij}(\pi_i, \pi_j) = 0$ **then**
    return $\langle \pi_1, \dots, \pi_n \rangle$;
  else
    report failure;
Function `Replan(`r, w`)`
  return trajectory π for robot r that minimizes $c(\pi) + w \sum_{j \neq r} \Omega_{rj}(\pi, \pi_j)$;

trates how penalty method resolves the corridor swap problem. Second, for many problem instances, the penalty method can find cheaper solutions than prioritized planning. For instance, if the penalty method is applied to the heads-on scenario (introduced in Figure 2.2.3), the algorithm constructs a solution in which both robots slightly divert their trajectories and effectively evenly dividing the collision avoidance effort. We note that this solution has a smaller total traveled distance than the solution returned by prioritized planning, i.e., a solution where only a single robot adjusts its trajectory to avoid the collision. Figure 7.3.3 illustrates how the penalty method resolves the heads-on problem.

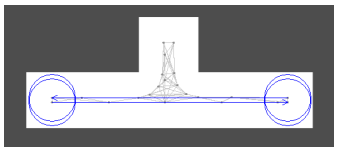
7.4 Experimental Evaluation

In this section, we compare the performance of our k-step penalty method (abbreviated as PM or PM(k=. . .)) against prioritized planning (PP) and a state-of-the-art optimal algorithm called operator decomposition (OD) [Standley, 2010] on a range of dense multi-robot path planning instances. Specifically, we focus on small and dense collision situations in which all robots are involved in a single conflict cluster. These situations usually represent bottlenecks in solving multi-robot pathfinding problems since sparser scenarios can be decomposed into small independent conflict clusters and solved separately.

Since reactive collision avoidance techniques based on the velocity obstacle paradigm [Fiorini and Shiller, 1998] are often used as a practical approach for collision avoidance between robots in multi-robot teams, we also compare our method with optimal reciprocal collision avoidance (ORCA).

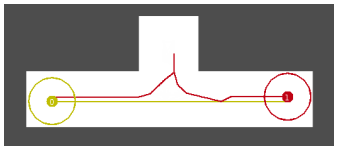
7.4.1 Scenarios

The experimental comparison is done in three scenarios depicted in Figure 7.4.1. The robots are modeled as 2-d discs that move on a 16-connected grid graph depicted in Figure 7.4.2. The start and goal position for each robot is chosen randomly. When generating start and destination for each robot we ensure that a) robots do not overlap at start position, b) robots do not overlap at goal position, and c) when adding a robot, its individually optimal trajectory must be in conflict with the trajectory of some previously added robot, i.e., the robots form a single conflict cluster. In Scenario A and B, the start and destination for each

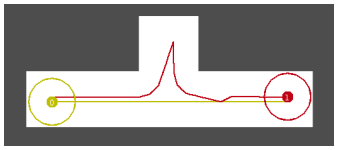


(a) The task is to swap positions: start and destination of each robot is depicted in blue. The graph used for trajectory planning is shown in gray.

(b) kPM: Initial trajectories



(c) kPM: Trajectories at iteration 3



(d) kPM: Final solution at iteration 9

For a video, visit: `http://youtu.be/HartJqN5HXM`

Figure 7.3.2: Corridor swap scenario: two circular robots have to swap their positions in the depicted narrow corridor. The frames show intermediate solutions generated by PM(k=10).

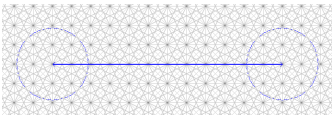
robot is chosen from the area depicted by the gray rectangle, however, the robots are allowed to leave the rectangle when resolving the conflict. In *Scenario C*, the robots must remain in the obstacle-free space when resolving the conflict. All robots move at the same maximum speed.

7.4.2 Experiment Setup

We generated 25 random problem instances for each team size and each environment. For each random instance, we run the following algorithms and recorded the runtime and the quality of the returned solution:

- **Penalty Method (PM):** The penalty-based method described in Section 7.3. The algorithm was run with different values of parameters k ranging from $k = 3$ to $k = 100$.
- **Prioritized Planning (PP):** The classical prioritized planning as described in Section 4.1. We used a fixed random priority order identical to the order used in the penalty method.
- **Operator Decomposition (OD):** Operator Decomposition [Standley, 2010] is a complete and optimal forward-search algorithm for multi-robot motion planning discussed in Section 2.2.2.2. We use OD algorithm to find the optimal cost for an instance.
- **Optimal Reciprocal Collision Avoidance (ORCA):** ORCA [van den Berg et al., 2011], is a popular reactive collision avoidance technique discussed in Section 2.2.1. In our implementation, we set the desired velocity vector such that it points at the shortest path to the destination, where the shortest path was computed on the same graph as used by the other methods. When using ORCA, we often witnessed dead-lock situations during which the robots either moved at an extremely slow velocity or stopped completely. If a prolonged deadlock situation was detected, we considered the run as failed.

Experiment runs were executed on 1 core of Intel Xeon E5-2665 2.40GHz, 4 GB RAM. A bundle that includes implementation of all the algorithms together with the problem instances can be downloaded at `https://github.com/mcapino/kpm-experiments`.



(a) The task is to swap positions: start and destination of each robot is depicted in blue. The graph used for trajectory planning is shown in gray.

(b) Solution found by prioritized planning. (Cost: 22.68)

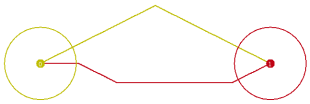
(c) kPM: Initial trajectories

(d) kPM: Trajectories at iteration 19

(e) kPM: Trajectories at iteration 20

(f) kPM: Trajectories at iteration 21

(g) kPM: Trajectories at iteration 23



(h) kPM: Final solution at iteration 30 (Cost: 21.84)

For a video, visit: `http://youtu.be/E7LFPIi3zhQ`

Figure 7.3.3: Heads-on scenario: the two robots need to swap their position in empty space. The frames show intermediate solutions generated by PM(k=20).

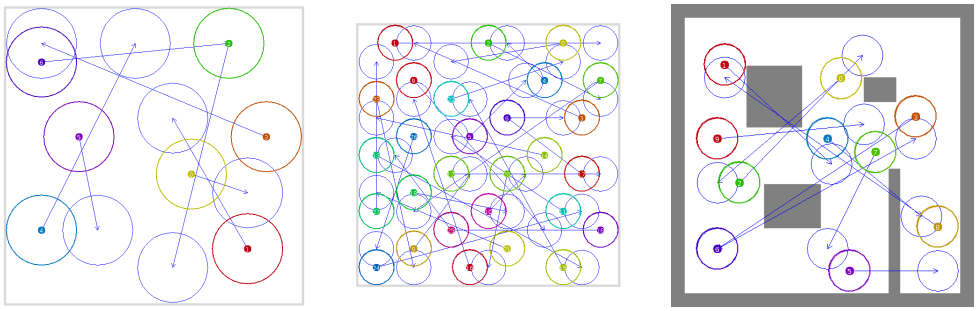
7.4.3 Results

In this section, we report on the results of the experimental comparison. We will discuss the success rate, solution quality, and the computational requirements of the compared algorithms:

Success rate The comparison of the success rate of the tested algorithms is in Figure 7.4.3a. The plots show the ratio of successfully solved instances for each algorithm in each test environment. Since the runtime of OD grows exponentially with the number of robots, we limited the runtime of OD to 1 hour. The simulation of ORCA was terminated with a failure if a deadlock was detected.

Sub-optimality Figure 7.4.3b shows average sub-optimality of solutions returned by the PM algorithm for different values of k parameter in each environment. The average suboptimality of solutions returned by PP algorithm is also shown for each environment. The sub-optimality of a solution for a particular instance is computed as $-(c - c^*)/c^*$, where c is the cost of the solution returned by the measured algorithm and $c^*$is the cost of optimal solution computed using OD algorithm. The averages are computed only from the subset of instances that were successfully resolved by all tested algorithms and for which we were able to compute the optimum value within 1 hour using the OD algorithm.

Time spent outside goal Figure 7.4.4a shows the average time spent outside goal position when the robots execute the solutions found by PM, PP and ORCA in all instances in Scenario B that involve 10 robots. The averages are computed on the subset of instances that were successfully resolved by all compared algorithms.

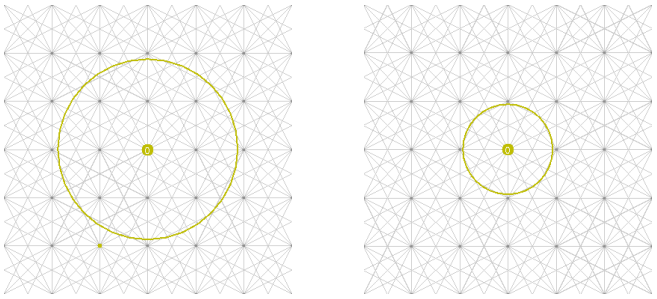


example instance, 7 robots (a) Scenario A

example instance, 25 robots (b) Scenario B

example instance, 10 robots (c) Scenario C

Figure 7.4.1: Experimental environments



(a) Discretization used in Scenario A

(b) Discretization used in Scenario B and C

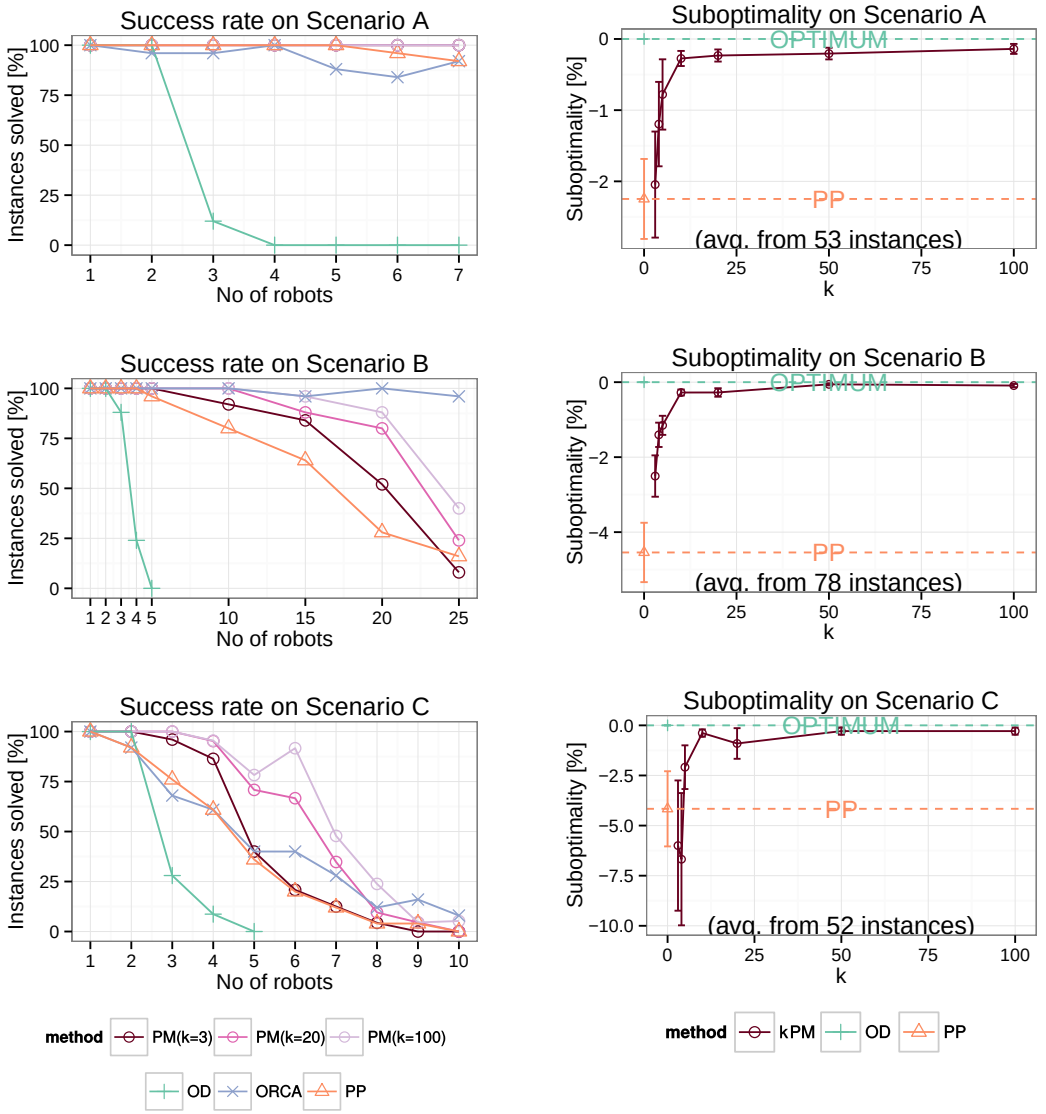
Figure 7.4.2: Discretization used for trajectory planning. The circle depicts the size of the body of a single robot.

CPU runtime Figure 7.4.4b shows average CPU runtime that PM and PP algorithms require to return a solution. The averages are computed only on the subset of instances that were successfully resolved by both PP and PM.

Results interpretation

Our results show that the optimal algorithm OD is practical only for instances that contain three or fewer robots. Prioritized planning generally scales better than OD, but returns solutions that are on average 2-4 % suboptimal (cf. Figure 7.4.3b). Although PM requires more time to provide a solution than PP (cf. Figure 7.4.4b), it does not exhibit the exponential drop in ability to solve larger instances that OD suffers from. Yet, the cost of solutions returned by PM tend to converge to the vicinity of the optimal cost with increasing number of iterations k for the instances where the optimum is known, (cf. Figure 7.4.3b) and otherwise converges to solutions that are of significantly lower cost than the solutions returned by PP (cf. Figure 7.4.4a). Further, besides the improved quality of the returned solutions, PM achieves higher success rate on our instances (cf. Figure 7.4.3a). E.g., in the empty environment with 20 robots (cf. Figure 7.4.3a), PM(k=100) solves 88 % of the instances, where PP solves only 28 %.

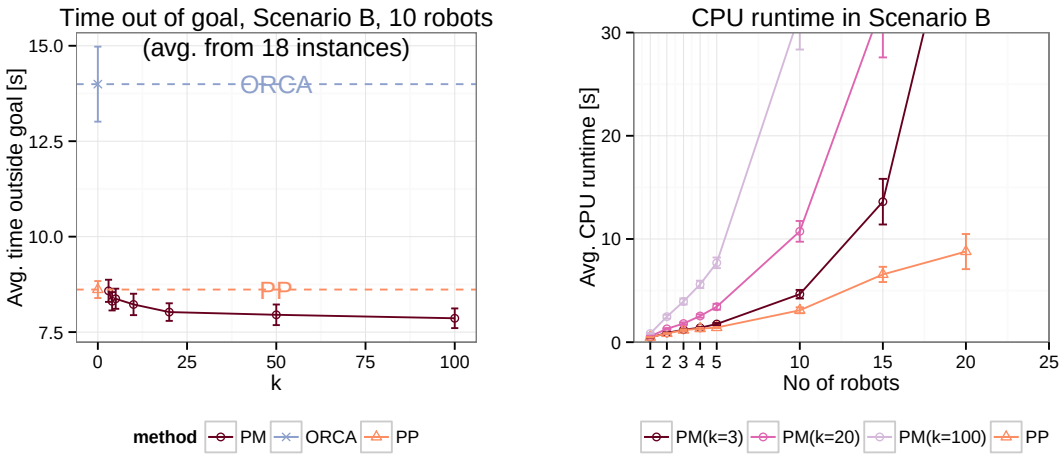
Although ORCA exhibits high success rate on our instances in Scenario B, it should be noted that the trajectories resulting from this collision avoidance process are very costly. We have observed that in instances involving a higher number of robots kPM returns trajectories that are more than 40 % faster than the trajectories returned by ORCA (cf. Figure 7.4.4a).



(a) Success rate. The plot shows the percentage of instances successfully solved by PM, PP, ORCA and OD in each environment. From PM, three different values of k parameter are considered: $k = 3$, $k = 20$, and $k = 100$.

(b) Optimality of penalty method. The plot shows average suboptimality of solutions returned by PM for different values of k parameter ranging from $k = 3$ to $k = 100$. The averages are computed on instances for which we were able to compute the optimum using the OD algorithm. The bars indicate the standard error of the average.

Figure 7.4.3: Success rate and solution quality



(a) Average time spent outside goal (i.e. cost). The plot shows average time the robots spend outside the goal if they execute solutions found by PM, PP and ORCA. For PM, we consider values of k ranging from $k = 3$ to $k = 100$. Measured on all instances with 10 robots in Scenario B that were successfully solved by all compared methods. The bars indicate the standard error of the average.

(b) CPU runtime requirements. The plot shows average CPU runtime needed by PM and PP to return a solution in instances involving different numbers of robots. From PM, three different values of k parameter are considered: $k = 3$, $k = 20$, and $k = 100$. The bars indicate the standard error of the average.

Figure 7.4.4: Time out of goal and CPU runtime comparison in Scenario B

Real-world maps

To demonstrate the applicability of our method, we have deployed the penalty method to coordinate trajectories of a number of simulated robots in two representative real-world environments. First, we tested the algorithm in an *office corridor* environment, which is based on the laser rangefinder log of Cartesium building at the University of Bremen.[1] The environment and the roadmap used for trajectory planning in the environment together with the task of each robot are depicted in Figure 7.4.5a. We run PM(k=5) algorithm to coordinate the trajectories of 16 robots sharing the environment and after 19 seconds obtained the trajectories shown in Figure 7.4.5b.

Second, we deployed the algorithm in a *logistic center* environment. The tasks of the individual robots and the roadmap that the robots used for planning in this environment are shown in Figure 7.4.6a. We used PM(k=5) algorithm to coordinate the trajectories of the individual robots. After 55 seconds we obtained coordinated trajectories shown in Figure 7.4.6b.

In this chapter, we have studied the applicability of penalty methods for multi-robot motion planning. We have investigated the question of the existence of separable optimal flow and consequently proposed a simplified adaptation of distributed optimization algorithm for multi-robot motion planning called k-step penalty method (kPM). We have experimentally analysed the ability of kPM algorithm to generate higher quality solutions and provide better instance coverage than prioritized planning. Our experimental results suggest that the proposed method can serve as a heuristic capable of generating near-optimal solutions while remaining tractable. In the next chapter, we will study the problem of executing coordinated trajectories in the presence of delaying disturbances.

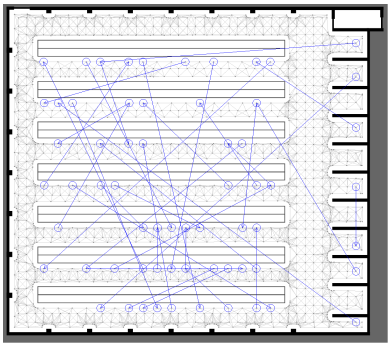
[1] We thank Cyrill Stachniss for providing the data through the Robotics Data Set Repository [Howard and Roy, 2003].

(a) Tasks of the robots and the roadmap used for planning

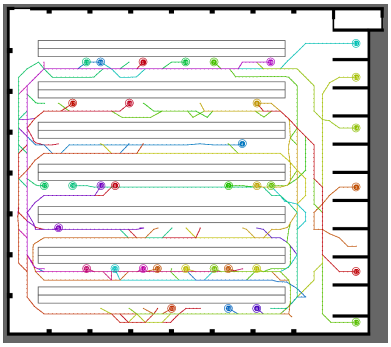
(b) Resulting coordinated trajectories.

For a video, visit: `http://youtu.be/VfiBuQBBIhM`

Figure 7.4.5: Office corridor: Coordination of 16 robots in an office corridor



(a) Tasks of the robots and the roadmap used for planning



(b) Resulting coordinated trajectories

For a video, visit: `http://youtu.be/G3A3TYKu73Q`

Figure 7.4.6: Logistic center: Coordination of 35 robots in a logistic center

Chapter 8

Executing Multi-robot Plans under Disturbances

The main drawback of the deliberative paradigm to multi-robot coordination, characterized by planning coordinated trajectories followed by execution of the computed plan, is that robots are guaranteed to reach their goals only if they follow the planned trajectories *precisely* both in space and time. Although the spatial component of the trajectory can be typically followed with reasonable tracking error using existing path tracking techniques, precise tracking of the temporal component is difficult to achieve in environments shared with humans due to the common requirement that the robots must yield to people, whose behavior is, however, hard to predict. Therefore, in mixed human-robot teams, the robots may be delayed by people who step in their way.

Observe that if one of the robots is delayed, and the others simply continue advancing along their trajectory, the robots may end up in a collision or a deadlock (see Figure 8.0.1 for an example). One possible approach to deal with such a disturbance is to find new coordinated trajectories from the current positions of all the robots to their destinations. Although replanning may sometimes work, it is usually too computationally intensive to be performed in a feedback loop and there is no guarantee that the resulting instance will be solvable in practical time. Another strategy, which we will refer to as ALLSTOP, is to stop all robots in the team every time any single robot is forced to stop. Although this strategy preserves liveness of the coordinated plan, it can be clearly very inefficient.

In this chapter, we show that stopping the entire team when a disturbance stops a single robot is in most situations unnecessary and provide a simple control rule specifying when a robot can proceed along its trajectory without risking future deadlocks or collisions. Therefore, the proposed method can be used to generalize guarantees of existing multi-robot trajectory planning techniques to environments, where robots might be forced to temporarily stop. Our simulation experiments further suggest that in such environments, executing preplanned trajectories using the proposed control rule is more reliable and more efficient than coordinating robots using reactive collision-avoidance techniques.

8.1 Problem Formulation

Consider a 2-d environment $\mathcal{W} \subseteq \mathbb{R}^2$ populated by n identical disc-shaped holonomic robots indexed $1, \ldots, n$. Their body has radius r and they can travel at unit maximum speed. Let $\pi_1, \ldots, \pi_n$ be feasible collision-free trajectories from the desired origin positions to the desired destination positions obtained from a multi-robot trajectory planner. The discrete-time trajectory of robot i is a function $\pi_i : \{0, \ldots, T\} \to \mathcal{W}$, where T denotes the time step when the last robot reaches its destination. The state of the system is described in terms of position $x_i \in \{0, \ldots, T\}$ of each robot i along its trajectory π_i, i.e., if a robot is in state x_i, then the robot is at spatial position $\pi_i(x_i)$. The control variables are

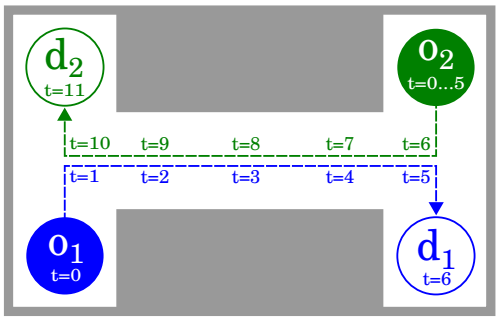


Figure 8.0.1: Example coordination problem. First robot desires to move from origin o_1 to destination d_1, the second robot desires to move from o_2 to d_2. A possible solution found by a multi-robot trajectory planner is indicated by the dashed line, where the planned position for each time point is annotated as $t = \ldots$ shown next to the trajectory. The plan dictates that the robot 1 should go through the corridor first. Robot 2 waits at its origin until robot 1 leaves the corridor and then proceeds through the corridor towards its destination. Now suppose that robot 1 is subject to a disturbance and enters the corridor with the delay of 5 time units. If robot 2 does not reflect on the delay and continues moving according to the original plan, both robots engage in a heads-on collision or deadlock, depending on whether some low-level safety mechanism prevents the robots from physically crashing, in the center of the corridor.

the advancement decisions for each robot, where the decision whether robot i should continue along its trajectory or stop at timestep t is denoted as $a_i(t) \in \{0, 1\}$. The advancement of each robot is subject to an exogenous multiplicative disturbance $\delta_i(t) \in \{0, 1\}$, where $\delta_i(t) = 0$ models the situation when the robot is forced to stop. Then, the discrete-time system dynamics is governed by the equation:

$$\forall t \in \mathbb{N} : \; x_i(t+1) = x_i(t) + a_i(t) \cdot \delta_i(t).$$

The state $x_i(t)$ of robot i can be intuitively interpreted as a position in the plan π_i at time t. Note that this position is measured in time units. The model of robot dynamic we use is further illustrated in Figure 8.1.1.

Our objective is to design a multi-robot controller $G(x_1, \ldots, x_n)$ that takes the current position of each robot and returns the advancement decision for the robots $(a_1, \ldots, a_n)$ such that from the initial state $\mathbf{x}_0 = (0, \ldots, 0)$ at time $t = 0$, the system is

- collision-free, i.e., the robots should never collide

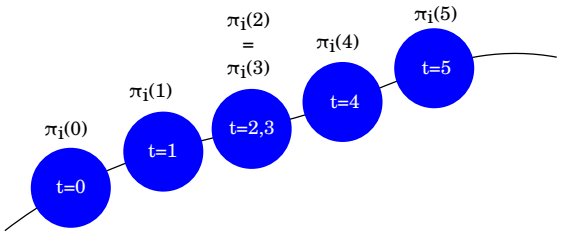
$$\forall t \, \forall i, j \; i \neq j : \quad |\pi_i(x_i(t)) - \pi_j(x_j(t))| > 2r,$$

- deadlock-free, i.e., the robots should eventually reach their destination

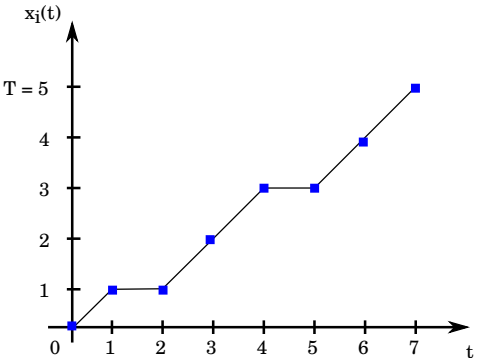
$$\forall i \, \exists t_f \, \forall t \geq t_f : \quad x_i(t) = T, \text{ and}$$

- efficient, i.e., disturbance affecting one robot should not lead to stopping the entire system.

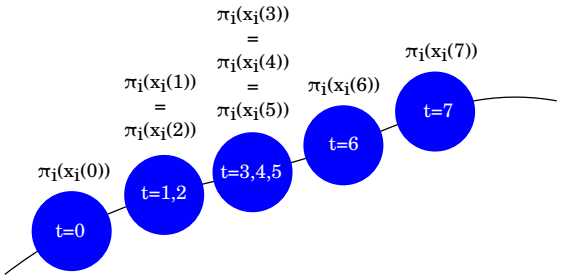
In the next section, we present a control scheme that satisfies the properties stated above. We remark that the presented scheme combines advantages of planning and reactive approach to multi-robot coordination in that it guarantees liveness and in the same time retains freedom of action allowing robots to deviate from the planned trajectory and handle the disturbance.



(a) Planned trajectory $t \mapsto \pi_i(t)$



(b) Robot state trajectory $t \mapsto x_i(t)$



(c) Actual trajectory $t \mapsto \pi_i(x_i(t))$

Figure 8.1.1: (a) An example output of the planner for robot i. The balls indicate the time evolution of the planned trajectory $t \mapsto \pi_i(t)$ for robot i. The curve represents the geometric path in $\mathbb{R}^2$ followed by robot i. (b) State trajectory $t \mapsto x_i(t) \in [0, 5]$ encodes the time evolution of the position of robot i along its planned trajectory. In our example, we can see that during time intervals $[1, 2]$ and $[4, 5]$ the robot makes no progress in its plan. Also, the fact that the state trajectory is below the diagonal can be interpreted as the robot lagging behind its plan. (c) The actual trajectory of the robot is $t \mapsto \pi_i(x_i(t))$. One can see that robot i will follow the same geometric path in $\mathbb{R}^2$ as planned. However, the time evolution along this geometric path is different. In particular, robot i will reach its goal at time 7 instead of time 5.

8.2 Control Scheme

In this section, we will introduce a simple control rule, which we will refer to as Robust Multi-Robot Trajectory Tracking Strategy (RMTRACK), that preserves safety and liveness of given multi-robot trajectory even under delay-inducing disturbances. To be able to concisely represent conditions about safe and unsafe mutual configurations of robots, we will make use of the coordination space formalism [O'Donnell and Lozano-Periz, 1989, LaValle, 2006]. Our approach relies on a transformation of the n-robot coordination problem in the physical space into the problem of finding a collision-free path in an n-dimensional abstract space called coordination space.

8.2.1 Coordination Space

A coordination space X for n robots is defined as an n-dimensional cube $X = \{0, \ldots, T\}^n$. A point in a coordination space encodes the state of all robots, i.e., the position along their trajectories. Let (x_i, x_j) be a point in a coordination space of two robots i and j with trajectories π_i and π_j. Then, the point (x_i, x_j) is said to be in collision denoted as $c_{ij}(x_i, x_j)$ if $|\pi_i(x_i) - \pi_j(x_j)| < 2r$. The set of all states in coordination space of robots i and j representing collision between the two robots is denoted as C_{ij} and defined as

$$C_{ij} := \{(x_i, x_j) \mid c_{ij}(x_i, x_j)\}\,.$$

Analogically, the collision region C in the coordination space of n robots is defined as

$$\begin{aligned} C &:= \{(x_1, \ldots, x_n) \mid \exists i, j \; i \neq j : \; c_{ij}(x_i, x_j)\} \\ &= \{(x_1, \ldots, x_n) \mid \exists i, j \neq i : \; (x_i, x_j) \in C_{ij}\} . \end{aligned}$$

By translating the original problem to the coordination space, our problem is now to design a controller that ensure that the trajectory of the multi-robot system in the coordination space $t \in \{0, \ldots, T\} \mapsto (x_1(t), \ldots, x_n(t))$ starting from $(0, \cdots, 0)$ will reach $(T, \ldots, T)$ in finite time t_f and at all times remains in collision-free region $X \setminus C$ of the coordination space X.

Observe that in the absence of disturbances the trajectory of the system in the coordination space will be a "diagonal" line segment connecting points $(0, \ldots, 0)$ and $(T, \ldots, T)$. Also, it is important to notice that the "diagonal" in the coordination space $t \in \{0, T\} \mapsto (t, \ldots, t) \in \{0, T\}^n$ is necessarily collision-free with respect to C since the planned trajectories $(\pi_1, \ldots, \pi_n)$ are assumed to be collision-free. Now, the problem is to design a control law that ensures that the collision region C is avoided even in the presence of disturbances.

8.2.2 Control Law

We now introduce RMTRACK control law that ensures that the actual trajectory in coordination space under disturbances avoids the collision region C, while simultaneously always making progress towards the point $(T, \ldots, T)$. The multi-robot control law $G(x_1, \ldots, x_n)$ is decomposed into a collection of control laws $\{G_i(x_1, \ldots, x_n)\}$, each governing the advancement of robot i. The control law for single robot i is defined as follows:

$$G_i(x_1, \ldots, x_n) := \begin{cases} 0 & \text{if } x_i = T \text{ or if } \big[\exists j : \; x_i > x_j \text{ and} \\ & \quad C_{ij} \cap (\{x_i + 1\} \times \{x_j, \ldots, x_i + 1\}) \neq \emptyset\big] \\ 1 & \text{otherwise.} \end{cases} \tag{8.2.1}$$

As we can see, the control law allows robot i to proceed only if the line segment from $(x_i + 1, x_j)$ to $(x_i + 1, x_i + 1)$ is collision-free in the coordination space X_{ij} for every other robot j. The mechanism is illustrated in Figure 8.2.1. The effect of the application of such a control law is shown in an example in Figure 8.2.2.

The control law allows robots to deviate from the planned trajectory in the coordination space but ensures that the actual trajectory the robots follow is homotopic to the planned trajectory, i.e., to the "diagonal" path. This is ensured by requiring that the trajectory in coordination space of any two different robots i and j stay at the same side of each connected component of C_{ij} as planned. Translated back to the original multi-robot formulation in $\mathbb{R}^2$ workspace, the above controller ensures that the robots will traverse overlapping parts of their geometric paths in the same order as implicitly specified in the planned trajectories.

8.3 Theoretical Analysis

In this section, we show that under certain technical assumptions, the RMTRACK control law satisfies collision-freeness and liveness properties, i.e., the robots are guaranteed not to collide and eventually reach their goal positions.

8.3.1 Collision-Freeness

In order to show collision-freeness, we make the following technical assumption. We assume that there is a 1-margin between the diagonal path and the obstacle region in the coordination space, i.e.,

$$\forall t \in \{0, \ldots, T-1\}, \; \forall i \neq j : \; \begin{cases} (t+1, t) \notin C_{ij} \\ \text{and} \\ (t, t+1) \notin C_{ij} \end{cases} \tag{8.3.1}$$

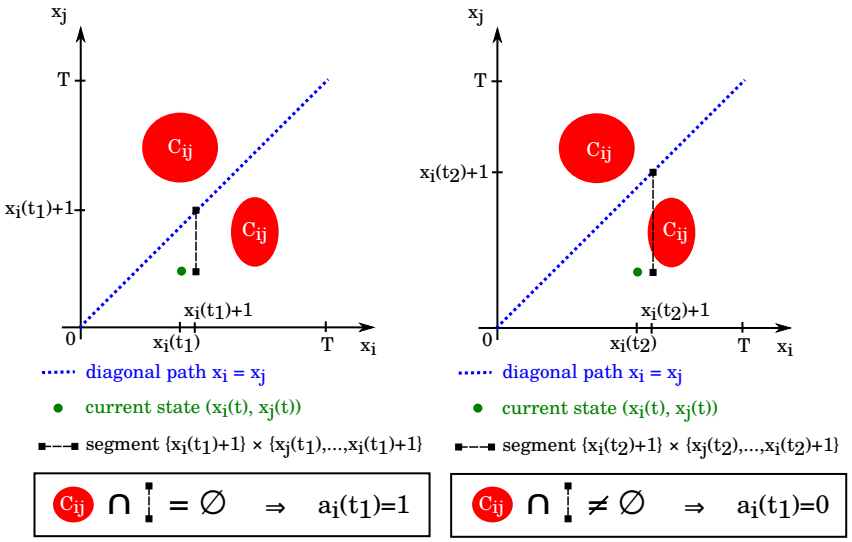


Figure 8.2.1: Illustration of the control law computation for robot i with respect to single other robot j. **Left:** The segment $\{x_i + 1\} \times \{x_j, \ldots, x_i + 1\}$ is collision-free with C_{ij} at time t_1, therefore the robot i is commanded to proceed, i.e., $a_i(t_1) = 1$. **Right:** The segment is not collision-free with C_{ij} at time t_2, therefore the robot i is commanded to stop, i.e., $a_i(t_2) = 0$.

This assumption is typically negligible as the geometric distance traveled by a robot in one time step is small. However, it can be satisfied by planning the reference trajectories with robots with slightly larger bodies.

Lemma 39. *Under control law G and assuming that Equation 8.3.1 holds, we have for all $t \in \mathbb{N}$ and for all $i, j \in \{1, \ldots, n\}$ s.t. $x_i(t) \geq x_j(t)$:*

$$C_{ij} \cap (\{x_i(t)\} \times \{x_j(t), \ldots, x_i(t)\}) = \emptyset \qquad (E_{i,j,t})$$

Proof. Initially, we have $x_1(0) = x_2(0) = \ldots = x_n(0) = 0$ and the state of robots does not belong to C_{ij}, so that $(E_{i,j,0})$ holds for all $i, j \in \{1, \ldots, n\}$.

Now, assume that $(E_{i,j,t})$ holds at some arbitrary time step $t \in \mathbb{N}$ for all $i, j \in \{1, \ldots, n\}$ s.t. $x_i(t) \geq x_j(t)$.

For each $i, j \in \{1, \ldots, n\}$, consider two options:

- If $x_i(t) = x_j(t)$, then $(E_{i,j,t}) \equiv (E_{j,i,t})$ hold and consider three options:
  - If $x_i(t+1) = x_j(t+1)$, $(E_{i,j,t+1}) \equiv (E_{j,i,t+1})$ will be satisfied as planned trajectories are collision-free.
  - If robot i moves one step forward while the other robot j is stopped, then by Equation 8.3.1, we have $(x_i(t+1), x_j(t+1)) = (x_i(t) + 1, x_i(t)) \notin C_{ij}$, and as planned trajectories are collision-free $(x_i(t+1), x_i(t+1)) \notin C_{ij}$, so that $\{x_i(t+1)\} \times \{x_j(t+1), x_i(t+1)\} \cap C_{ij} \neq 0$, so that $(E_{i,j,t+1})$ is satisfied with $x_i(t+1) > x_j(t+1)$.
  - If robot i moves one step forward while the other one j is stopped,, we use the symmetric reasoning to obtain that $(E_{j,i,t+1})$ holds.
- If $x_i(t) > x_j(t)$ then we have $x_i(t+1) \geq x_j(t+1)$ and consider three options:
  - If neither of robots moves, $(E_{i,j,t+1})$ will still be obviously satisfied as $(E_{i,j,t+1}) \equiv (E_{i,j,t})$ which holds.

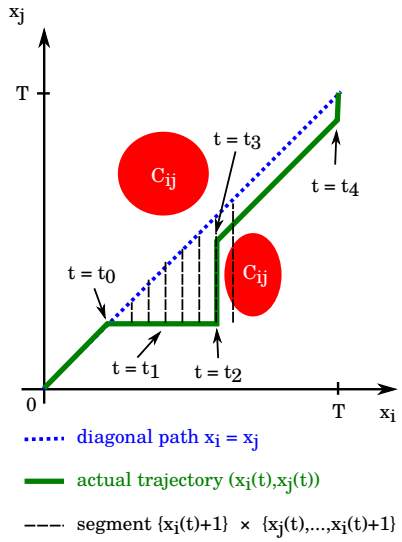


Figure 8.2.2: Trajectory of the robots in the coordination space under control law G. At time $t = t_0$, a disturbance makes robot j stop ($\delta_j(t_0) = 0$). The system leaves the diagonal path, and the control law allows robot i to proceed until time $t = t_2$ where robot i is commanded to stop, i.e., $a_i(t_2) = 0$ (see Figure 8.2.1). Finally, disturbance for robot j goes away, and robot j proceeds. It is only at time $t = t_3$ that the control law allows robot i to proceed. Robot i reaches its goal at time $t = t_4$ and robot j right after. No collision occurred and note how the control law ensured that the actual trajectory in the coordination space remains on the same side as the diagonal path with respect to each connected component of the obstacle region. This is what enables to guarantee liveness, as changing the homotopy class may lead to a deadlock configuration with respect to other robots.

– If robot i does not move, then we have:

$$\begin{aligned}&\{x_i(t+1)\} \times \{x_j(t+1), \ldots, x_i(t+1)\}\\ &= \{x_i(t)\} \times \{x_j(t+1), \ldots, x_i(t)\}\\ &\subseteq \{x_i(t)\} \times \{x_j(t), \ldots, x_i(t)\}\end{aligned}$$

which does not intersect C_{ij} as $(E_{i,j,t})$ holds, so that $(E_{i,j,t+1})$ is satisfied.

– If robot i moves, then by construction of G and because $x_i(t) > x_j(t)$, we have:

$$C_{ij} \cap (\{x_i(t)+1\} \times \{x_j(t), \ldots, x_i(t)+1\}) = \emptyset \tag{8.3.2}$$

Taking into account that $x_i(t+1) = x_i(t) + 1$ and $x_j(t+1) \in \{x_j(t), x_j(t)+1\}$, we obtain:

$$\begin{aligned}&\{x_i(t+1)\} \times \{x_j(t+1), \ldots, x_i(t+1)\}\\ &= \{x_i(t)+1\} \times \{x_j(t+1), \ldots, x_i(t)+1\}\\ &\subseteq \{x_i(t)+1\} \times \{x_j(t), \ldots, x_i(t)+1\}\end{aligned}$$

which does not intersect C_{ij} by Equation 8.3.2, so that $(E_{i,j,t+1})$ holds.

By induction, we conclude that $(E_{i,j,t})$ is satisfied for all $t \in \mathbb{N}$ and $i, j \in \{1, \ldots, n\}$ s.t. $x_i(t) \geq x_j(t)$. □

Theorem 40. *Under control law G and assuming that Equation 8.3.1 holds, the trajectory in the coordination space is collision-free, i.e.,*

$$\forall t \in \mathbb{N}: \ (x_1(t), \ldots, x_n(t)) \notin C.$$

Proof. Take an arbitrary time step $t \in \mathbb{N}$. Assume that $(x_1(t), \ldots, x_n(t)) \in C$. Then, there exists i, j such that $(x_i(t), x_j(t)) \in C_{ij}$ and we can assume without loss of generality that $x_i(t) \geq x_j(t)$. This is in contradiction with Lemma 39. □

8.3.2 Liveness

First, we characterize our assumptions on disturbances. Clearly, it is possible to construct a disturbance function that will prevent the system from reaching configuration $(T, \ldots, T)$ under any control law. For example, if for a given robot $i \in \{1, \ldots, n\}$, we have $\forall t \in \mathbb{N}, \ \delta_i(t) = 0$, then it is impossible for robot i to reach its goal. Therefore, in the following analysis, we assume that disturbances do not prohibit any of the robots from reaching its goal, i.e., we consider systems in which disturbances may delay any given robot for arbitrarily long, but the robot will eventually be able to reach the goal. Formally, we say that disturbances are *non-prohibitive,* if for any controller that satisfies

$$\forall t \in \mathbb{N}: \begin{cases} x_1(t) = x_2(t) = \ldots = x_n(t) = T \\ \text{or} \\ \exists i \in \{1, \ldots, n\}: \ x_i(t) < T \text{ and } a_i(t) = 1 \end{cases},$$

the system will eventually reach the goal, i.e there exists $t_f \in \mathbb{N}$ such that $(x_1(t_f), \ldots, x_n(t_f)) = (T, \ldots, T)$. In other words, as long as the controller lets at least one unfinished robot proceed at any point of time, all robots will eventually reach their goal. The time of goal achievement t_f might be affected by disturbances, but disturbances will not prevent goal achievement in finite time.

Under *non-prohibitive* disturbances, the control law G guarantees liveness:

Lemma 41. *Under control law G, there is at least one robot proceeding at any point of time, i.e.,*

$$\forall t \in \mathbb{N}: \begin{cases} x_1(t) = x_2(t) = \ldots = x_n(t) = T \\ \textit{or} \\ \exists i \in \{1, \ldots, n\}: \ x_i(t) < T \textit{ and } a_i(t) = 1 \end{cases}$$

Proof. Define $I(t) \subseteq \{1, \ldots, n\}$ as follows:

$$I(t) := \arg\min_i \ x_i(t) = \left\{ i \,|\, \forall j \neq i: \quad x_j(t) \geq x_i(t) \right\}$$

$I(t)$ exists and is a non-empty set as $\{1, \ldots, n\}$ is finite. By construction of G, we have

$$\forall i \in I(t): \quad a_i(t) = G_i(x_1(t), \ldots, x_n(t)) = 1 \text{ or } x_i(t) = T.$$

There are two scenarios: a) Either for all $i \in I(t)$, $x_i(t) = T$, then $\min_i x_i(t) = T$, so that we have $x_1(t) = x_2(t) = \ldots = x_n(t) = T$. b) Or there exists some $i \in I(t)$ such that $x_i(t) < T$ and $a_i(t) = G_i(x_1(t)x_n(t)) = 1$. □

Theorem 42. *Control law G ensures liveness under non-prohibitive disturbances, i.e.,*

$$\exists t_f: \ x_1(t) = x_2(t) = \ldots = x_n(t) = T.$$

Proof. This is a direct consequence of the preceding lemma and the assumption made on disturbances. □

Empty hall with 25 robots | Office corridor with 25 robots | Warehouse with 30 robots

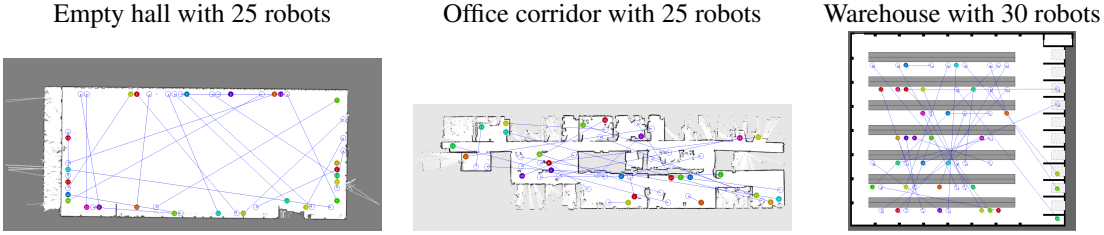


Figure 8.4.1: Maps used for experimental comparison. The figures show an example problem instance in each environment. The filled circles represent robots. The arrows indicate the desired destination of each robot.

8.4 Experimental Evaluation

In this section, we will discuss the results of the experimental comparison of RMTRACK approach against the baseline liveness-preserving method ALLSTOP and a reactive method ORCA using multi-robot simulation.

Experiment Setup

The comparison was performed in three environments: Empty hall, Office corridor and Warehouse as shown in Figure 8.4.1. A single problem instance in one of the environments consists of n robots attempting to move from randomly generated origins to randomly generated destinations. We first find collision-free multi-robot trajectories from the origins to the destinations and then let each robot follow the given trajectory while randomly disturbing its advancement with specified intensity. More precisely, every second we decide with probability corresponding to the disturbance intensity whether during the next second the robot will be prevented from moving.

To ensure that the initial coordinated trajectories can be found in a reliable and tractable fashion, the test environments are designed to be well-formed infrastructures, which allowed us to use revised prioritized planning approach to efficiently find the initial trajectories for the robots to follow.

In Empty hall and Warehouse environments, we generated 10 instances with 10 robots and 10 instances with 50 robots; In Office corridor environment, we generated 10 instances with 10 robots and 10 instances with 35 robots. Note that a single instance represents a specific assignment of origins and destinations to the robots.

The source code of all algorithms and benchmark instances are available at `https://github.com/mcapino/rmtrack`. For an illustrative video, visit `https://youtu.be/29YRLJpB9OQ`.

Comparison of RMTRACK and ALLSTOP

First, we compare two liveness-preserving control laws to handle disturbances: 1) ALL-STOP: the baseline law that makes entire multi-robot team stop whenever a single robot is disturbed and 2) RMTRACK: the law proposed in Section 8.2.2. For each instance and disturbance intensity ranging from 0 % to 50 %, we run both algorithms and measured the time it took for each robot to reach its destination.

In order to isolate the effect of each control law on the travel time from the effect of disturbances and the effect of the quality of the initial plan, we compute a lower bound on the travel time of each robot assuming fixed disturbance and fixed initial plan. Such a lower bound is obtained by simulating the robot such that the inter-robot collisions are ignored and thus all robots always command to proceed at maximum advancement rate along their initial trajectory. Then the average advancement rate of the robot and consequently the travel time is affected solely by disturbances. In fact, for uniformly distributed random disturbance with equal intensity q for all robots, this corresponds to robots advancing

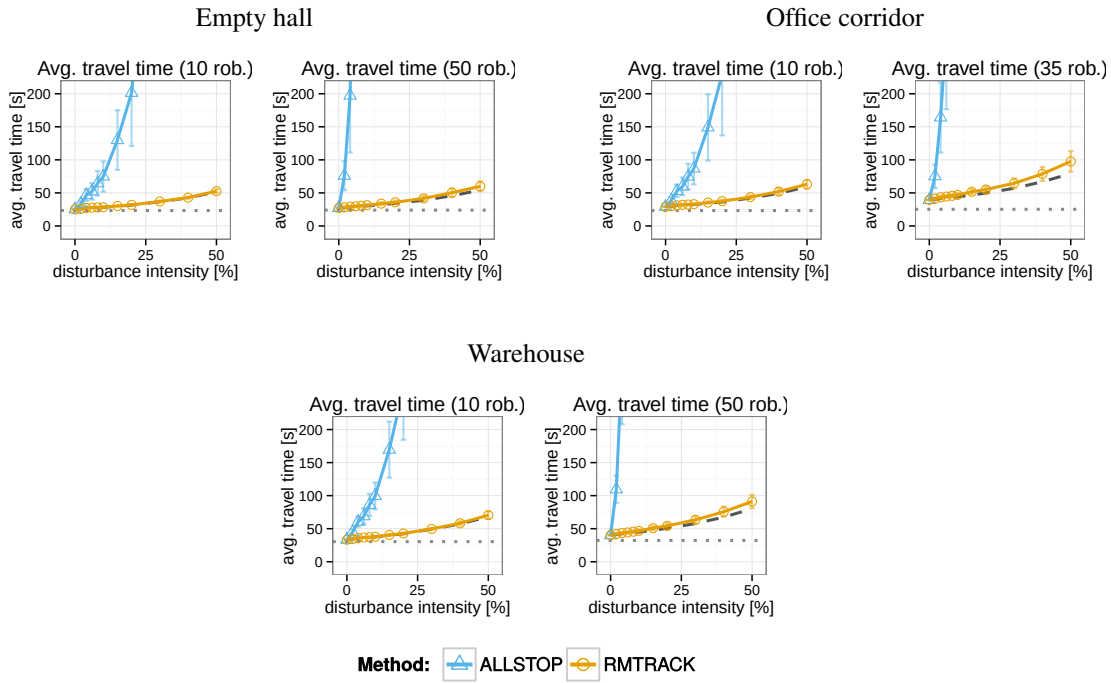


Figure 8.4.2: Experimental comparison of ALL-STOP strategy with RMTRACK. Each datapoint is an average travel time of a single robot from its origin to its destination under the given disturbance intensity using one of the two evaluated control strategies. The dashed line represents the average lower bound on the travel time under the given disturbance intensity. The dotted line representes the average travel time from origin to destination assuming no disturbance and no need for coordination between robots. The bars represent standard deviation of the difference between the travel time under the evaluated algorithm and the lower bound travel time.

on expectation at $1 - q$ fraction of the original advancement rate 1. Thus, the lower bound on expected travel time under disturbance intensity q can be also computed as $E(t_f)/(1 - q)$, where $E(t_f)$ denotes expected travel time in the absence of disturbance.

It is easy to see that this lower bound represents the best possible travel time that can be achieved by RMTRACK, for instance, the travel time when the paths of the robots do not overlap. On the other hand, it is not difficult to construct a combination of problem instance and disturbances for which the behavior of RMTRACK degenerates to that of ALLSTOP. Curiously, since ALLSTOP proceeds only when none of the robots is disturbed, which for uniformly distributed disturbance with intensity q at each robot happens with probability $(1 - q)^n$, the expected travel time for ALLSTOP strategy can be consequently computed as $E(t_f)/(1 - q)^n$, where $E(t_f)$ is again the expected travel time without disturbance and n is the number of robots in the system.

Consequently, we expect the average traveltime under RMTRACK strategy to be bounded from below by the lower-bound travel time and by the ALLSTOP travel time from above. The actual travel time under RMTRACK will then depend on the "interdependency" of initial trajectories and the level of disturbance. Given these two bounds, an interesting question is how will RMTRACK strategy perform in characteristic real-world environments.

The results of performance comparison of RMTRACK with respect to ALLSTOP and the lower bound travel time for the three test environments are shown in Figure 8.4.2. We can see that consistently over all test environments and for different numbers of robots, the baseline strategy ALLSTOP quickly becomes impractical when the disturbance intensity is high. In contrast, the average travel time under

RMTRACK remains reasonable even for high intensities of disturbance. Further, it is encouraging that for all three environments we tested on, the average travel time under RMTRACK remains close to the lower-bound travel time.

Comparison of RMTRACK and ORCA

Next, we compared RMTRACK strategy with a reactive collision-avoidance technique ORCA [van den Berg et al., 2011]. In our implementation, the desired velocity at each time instance follows the shortest path to destination. For each instance we run ORCA and RMTRACK techniques for different disturbance intensities ranging from 0 % to 50 %. During the experiment, we often witnessed ORCA leading robots to dead-lock situations during which the robots either moved at extremely slow velocities or even stopped completely. Therefore, if the robots failed to reach their destination within 10 minutes[1], we considered the run as failed.

Figure 8.4.3 summarizes the results of the comparison. We can see that the success rate of ORCA deteriorates with increasing disturbance intensity. This is perhaps surprising since reactive methods are believed to be particularly well suited for unpredictable environments. Among the reasons behind this phenomena seems to be that the reciprocal reactive algorithms rely on all robots executing the same algorithm and consequently on "splitting" the collision avoidance effort. This assumption is however violated if one of the robots is disturbed and does not execute the velocity command that the algorithm computed. The plots in the bottom row show comparison of performance of RMTRACK and ORCA. We can see that even when ORCA solves a given instance, the expected travel time for a robot is on expectation longer, especially so in cluttered environments and for high disturbance intensities.

In this chapter, we studied the problem of executing coordinated trajectories in multi-robot system affected by delaying disturbances. We have proposed a simple control rule that controls the advancement of individual robots in the case when some of the robots are delayed. While naive strategies may cause deadlocks, we have shown that the proposed strategy guarantees liveness of the system, i.e., the robots are guaranteed to avoid all deadlocks and reach their goals. Besides being deadlock-free, the application of the proposed control rule allows robots to reach their goals faster than the application of a reactive collision avoidance technique. In the next chapter, we will report on the results of the deployment of ADPP algorithm in two real-world multi-robot systems.

[1] average travel time between origin and destination ignoring collisions and without disturbance is around 25 second

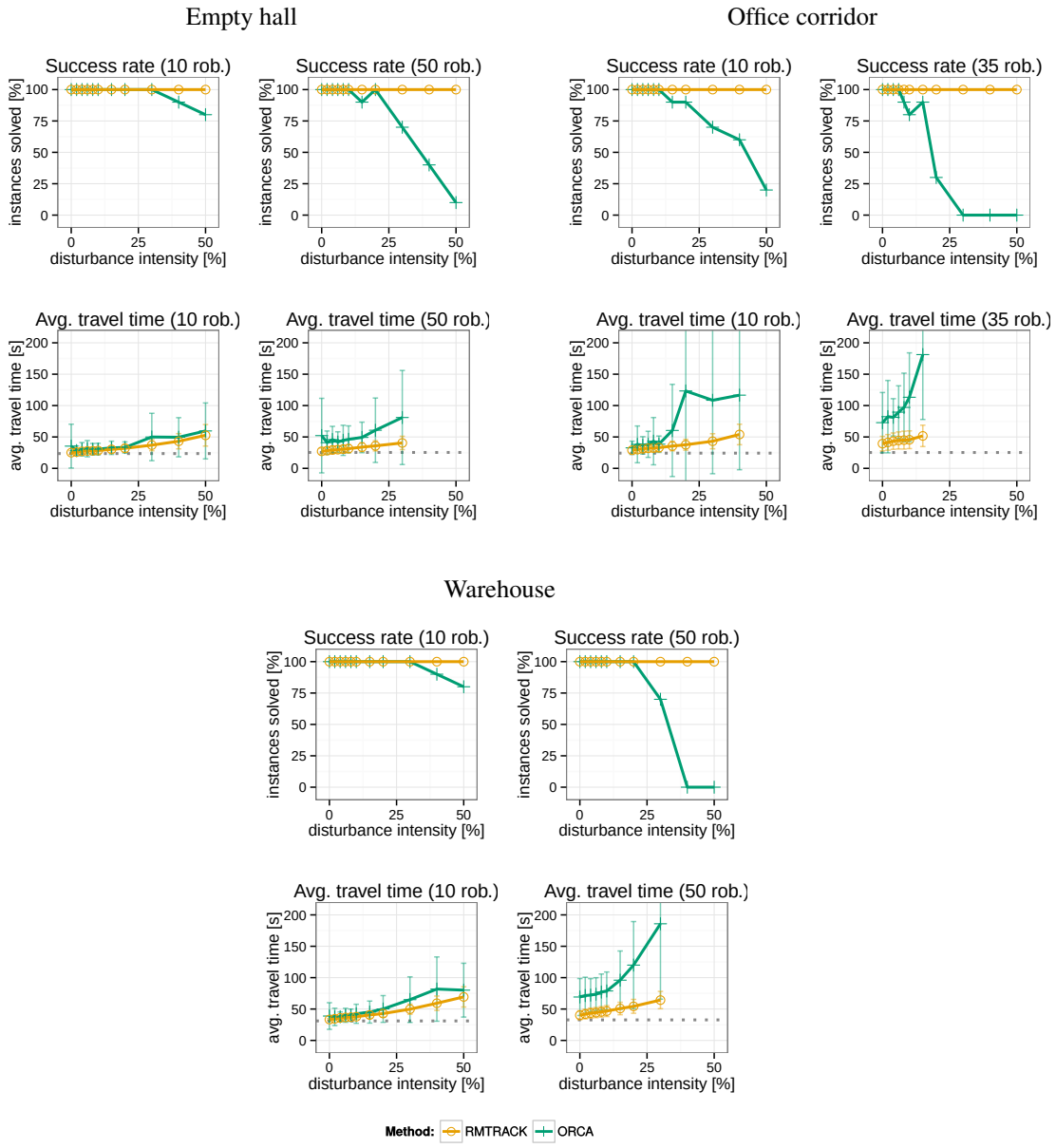


Figure 8.4.3: Experimental comparison of ORCA and RMTRACK. The average travel time is computed if there are at least five instances for given number of robots and disturbance intensity successfully solved by both evaluated algorithms. The dotted line represents the average travel time from origin to destination assuming that no disturbance and no need for coordination with other robots. The bars represent the standard deviation of the difference between the travel time under the evaluated algorithm and the average travel time from origin to destination ignoring collisions with other robots and disturbance.

Chapter 9

Deployments

In this chapter, we demonstrate the practical applicability of the coordination paradigm introduced in the preceding chapters by field deploying it as a collision avoidance mechanism in a non-trivial multi-robot system. More specifically, we deployed the asynchronous decentralized prioritized planning (ADPP) algorithm as a conflict resolution mechanism in an experimental multi-UAV system and an experimental mobility-on-demand system utilizing self-driving vehicles. The first mentioned multi-robot system has been developed at CTU in Prague by Selecký, Štolba, Meiser, Čáp, Komenda, Rollo, Vokřínek, and Pěchouček [2013] and consists of a team of cooperating fixed-wing UAVs designed to autonomously carry out tasks such as patrolling, target tracking, or area surveillance. The second system has been developed at Singapore-MIT Alliance for Research and Technology (SMART) research institute by Chong, Qin, Bandyopadhyay, Wongpiromsarn, Rebsamen, Dai, Kim, Ang, Hsu, Rus, and Frazzoli [2012] and consists of a fleet of autonomous golf carts designed to provide autonomous personal transportation within a part of NUS campus in Singapore. In the following, we describe each system in detail, explain the necessary adaptations to the ADPP algorithm, and discuss the results of the deployments in both multi-robot systems.

9.1 Multi-UAV Testbed

The Multi-UAV testbed is an experimental multi-robot system that consists of two hardware UAVs and an arbitrary number of simulated UAVs. The hardware UAVs are based on the Unicorn airframe equipped with the Kestrel Autopilot from Lockheed Martin (see Figure 9.1.1). Further, the airborne system is equipped with Gumstix Overo EarthStorm embedded computer for on-board computation and Xbee 2.4GHz RF module to enable direct UAV-to-UAV messaging. Due to the limited number of physical UAVs at our disposal, a mixed-reality simulation approach [Jakob et al., 2012] is used to evaluate the behavior of tested multi-agent coordination algorithms in systems with more than two

Figure 9.1.1: Left: Unicorn fixed-wing UAV from Lockheed Martin. Right: One of UAVs during the field experiment.

Figure 9.2.1: Autonomous golf cart

UAVs. To account for a limited precision of plan execution due to unstable wind, we model the UAV for the purposes of the trajectory planning as a large cylindric zone around each UAV. Specifically, we set the radius of the cylinder to 100 m and the half-height of the cylinder to 10 m. The parameters were chosen empirically and reflect the observed precision of trajectory tracking in difficult wind conditions. An individual UAVs can communicate using a radio link having the shared capacity 5 kBps in ideal conditions. To keep the latency low and to use the bandwidth efficiently, the data are transferred in a raw form and thus the data delivery is not guaranteed.

The system is designed to allow testing of multi-UAV applications such as cooperative area surveillance or cooperative target tracking. Often, the trajectories of different UAVs executing their assigned tasks would not have enough separation to ensure safe collision-free execution, i.e. the trajectories of the two UAVs are in a conflict. Therefore, a motion coordination mechanism was needed to resolve the conflicts between the trajectories of individual UAVs.

9.2 AMoD System

The experimental Automated Mobility on Demand (AMoD) System consists of a small fleet of autonomous golf carts shown in Figure 9.2.1. The vehicle platform is a standard off-the-shelf golf cart Yamaha G22E with the driving wheel removed and equipped with additional sensors, actuators, and computers. The sensors installed on the car include two forward facing and two backward facing planar LIDARs, two cameras, an inertial measurement unit and wheel encoders. The car is actuated using a stepping motor attached to a steering wheel axle, a push-pull mechanism that controls the brake pedal and by controlling the throttle signal. Due to a limited precision of GPS localization in urban environments, the vehicle uses the monte-carlo particle-based localization primarily relying on the laser scan data from the LIDARS. The dynamic obstacles such as pedestrians are detected by fusing the information from LIDAR and the front facing camera. In result, each vehicle is capable to reliably localize itself in a known map, follow a given path, and automatically stop whenever an obstacle appears close to its planned path. For the purposes of motion coordination, the vehicle was equipped with Cohda MK2 wireless communication device based on 802.11p vehicular communication standard that allows individual vehicles to exchange messages.

The system was designed to provide autonomous transportation within a small part of NUS campus. The passengers can summon a vehicle and command it to drive to any station in the system. It is possible, however, that the planned paths of the vehicles traveling to different stations overlap and thus vehicles need a mechanism for coordinating their trajectories to avoid deadlocks or potential collisions.

9.3 Closed-Loop AD-PP

In both multi-robot systems, we needed a decentralized method that will ensure that the trajectories of the robots remain coordinated. The task could be achieved by decentralized prioritized planning, but the algorithm had to be adapted to address the specifics of the target real-world multi-robot systems, namely:

1. In both systems, the whole team or individual robots may be re-tasked by an operator and thus the goals of any of the robots may change at any time during the mission execution.

2. The trajectories generated by the planner are often executed imprecisely by the robot. In the multi-UAV systems, the execution can be poor due to unstable wind, while in the AMoD system the vehicles may be delayed by pedestrians.

3. The radio communication channel used in both systems does not guarantee message delivery and message loss may occur.

4. During the mission execution, some robots may be removed from the robotic team, while some new ones can be added.

To address these requirements, we implemented the asynchronous decentralized prioritized planning scheme to work in a closed-loop fashion. In these specific systems, a decentralized implementation of classical prioritized planning turned out to be a better fit than a decentralized implementation of revised prioritized planning: First, since robots can replan at any time from any point in space to react to disturbance, we are not able to guarantee that $S^{>i}$-avoiding and $G^{<i}$-avoiding satisfying path will always exists, thus neither of the algorithms can guarantee completeness. Second, in situations similar to the one depicted in Figure 4.3.4, which are not uncommon in our case, PP returns shorter solution than RPP. Therefore, we chose to use PP over RPP as an underlying planning scheme. Further, we chose asynchronous decentralized implementation of PP over the synchronized implementation, because in SD-PP the robots would need to run some form of distributed termination detection algorithm [e.g. Mattern, 1987] at the end of each round in order to detect that every other robot has finished computing in the given round, which adds extra complexity to the algorithm. Moreover, with unreliable communication channels, such synchronization is fundamentally impossible to achieve with certainty [Akkoyunlu et al., 1975].

We note that that the more recent techniques RMTRACK and COBRA presented in Chapters 6 and 8 are able to handle delaying disturbance, incrementally assigned tasks, and addition/removal of robots from the system in runtime while retaining completeness guarantees, however these techniques were not fully developed when the two experiments were conducted and are therefore not utilized.

The pseudocode of the closed-loop version of the AD-PP (called CLAD-PP) is given in Algorithm 12. In CLAD-PP, the execution of the trajectories is continuously monitored and a replanning is invoked if necessary. The replanning is triggered: a) if the robot is assigned a new task (e.g. by an operator); or b) if the robot has diverted from its planned trajectory.

Observe that each such forced replanning triggers a new trajectory coordination query for the lower-priority robots and thus it may cause a cascade of replannings for lower-priority robots. However, just as in the standard AD-PP, each of the robots will eventually adapt with a conflict-free trajectory or reports a failure to find one. Because of the possible message loss, the UAVs broadcast their planned trajectory not only when it changes, but also periodically onwards. The possible dynamic team reconfiguration

does not allow to wait for a global termination of the ADPP run, and therefore, a robot starts executing its collision-free trajectory immediately when it is planned (lines 5, 12, and 15). For the trajectory generation (BESTTRAJ routine) in multi-UAV system we use an any-time RRT*-based [Karaman and Frazzoli, 2011] spatio-temporal trajectory planner that is terminated after one second, while in the AMoD system we use a time-extended roadmap planner (see Section 4.4) over a hand-designed roadmap of possible paths among stations in the systems.

From a practical standpoint, it should be noted that such a closed-loop implementation is possible due to the asynchronous nature of the AD-PP approach. Implementing such a dynamic mechanism with a centralized or a synchronized algorithm would be much less natural.

Algorithm 12: Asynchronous Decentralized Implementation of Prioritized Planning, Closed-Loop version

1 **Algorithm** `CLAD-PP`
2 $\quad \pi_i \leftarrow \emptyset$;
3 $\quad H_i \leftarrow \emptyset$;
4 $\quad \pi_i \leftarrow$`Find-consistent` $(\pi_i, \mathcal{W}, \Delta(H_i))$;
5 $\quad$ **follow** π_i;
6 **Handle-message** $\mathrm{INFORM}(j, \delta_j)$
7 $\quad$ **if** $j < i$ **then**
8 $\quad\quad H_i \leftarrow \left(H_i \setminus \left\{(j, \delta'_j) \,:\, (j, \delta'_j) \in H_i\right\}\right) \cup \{(j, \delta_j)\}$;
$\quad$ `// plans from the robot's current position`
9 $\quad \pi_i \leftarrow$ `Find-consistent` $(\pi_i, \mathcal{W}, \Delta(H_i))$;
10 $\quad$ **if** $\pi_i = \emptyset$ **then**
11 $\quad\quad$ report failure and terminate;
12 $\quad$ **follow** π_i;
13 **When** *task changes* ***or*** *robot diverted from* π_i
$\quad$ `// plans from the robot's current position`
14 $\quad \pi_i \leftarrow$`Find-consistent` $(\pi_i, \mathcal{W}, \Delta(H_i))$;
15 $\quad$ **follow** π_i;
16 **Periodically**
17 $\quad$ **broadcast** $\mathrm{INFORM}(i, R_i^{\Delta}(\pi))$;

9.4 Results

In this section, we discuss our experience with deployment of CLAD-PP algorithm in Multi-UAV and AMoD systems.

Multi-UAV Testbed

We used CLAD-PP as a default collision avoidance method within Multi-UAV system. It was regularly field tested in the air and provided satisfactory results. Whenever any of the UAVs generated a trajectory that conflicted with trajectories of other UAVs in the air, the affected robots instantly recomputed their trajectories to avoid the collision. Most of the time, the process of trajectory deconfliction took less than a second.

A popular collision avoidance benchmark that we will use to demonstrate the behavior of the proposed technique is the "superconflict" scenario. In the superconflict scenario, there are four UAVs with

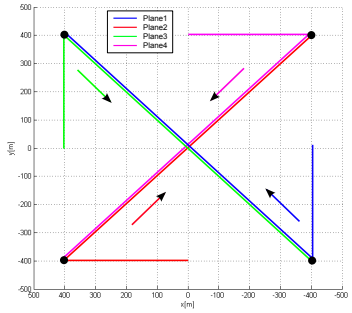
starting positions placed at the corners of a square and their goals being at the respective diagonally opposite corners. Hence, all the airplanes are initially in a conflict in the middle of the square, see Figure 9.4.1a. In order to show the behavior of the CLAD-PP technique more clearly, the BESTTRAJ trajectory planners of the individual UAVs were constrained to generate trajectories in fixed altitude with fixed velocity. In this experimental setup, two real UAVs (Plane1 and Plane2) are attached to a hardware-in-the-loop simulator and the two others (Plane3 and Plane4) are simulated. The control algorithms are deployed and run on the Gumstix onboard computers. The hardware UAVs use the safe zone radius 110 m, while the simulated ones use 70 m. The virtual UAVs are controlled by the identical software as the hardware UAVs; however, they run in independent virtual machines on a desktop computer. Both real and simulated UAVs communicate via their XBee radio modules. The Kestrel autopilot of the hardware UAVs is connected to a high-fidelity flight simulator Aviones[1] in the hardware-in-the-loop mode. When the mission is started, the UAVs execute the CLAD-PP algorithm to coordinate their motions. The resulting traces that were recorded during the experiment are shown in Figure 9.4.1b and in the linked video. One can see that the solution has a structure typical for prioritized planning – the highest-priority Plane 1 keeps its first straight-lane trajectory, while all other UAVs need to change their initial trajectory to adapt.

AMoD System

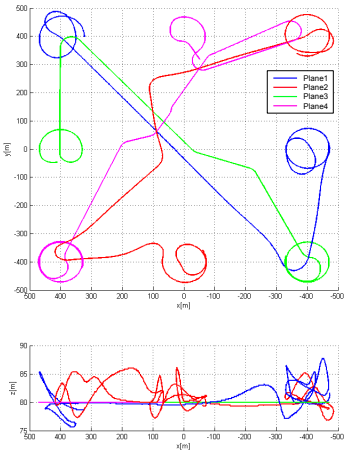
The CLAD-PP algorithm was successfully integrated into the ROS-based software stack of the autonomous golf carts to provide a mechanism for trajectory coordination between individual vehicles. The behavior of the algorithm was tested with two vehicles in a number of scenarios. The vehicles were ordered to move from their current position to another station such that the trajectories of the two vehicles are at some point in collision. In all cases the algorithm exhibited satisfactory behavior, i.e., the lower-priority vehicle replanned its trajectory to a collision-free alternative in less than three seconds after the conflict between the robots was induced. Figure 9.4.2 shows three selected scenarios in which we tested the algorithm, pictures from the field experiment and a link to an illustrative video from the experiment.

In this chapter, we have reported on the results of a successful deployment of a closed-loop version of ADPP algorithm for trajectory coordination in two real-world multi-robot systems: a multi-UAV system and an automated mobility-on-demand system. The next chapter will conclude the thesis.

[1] http://aviones.sourceforge.net/



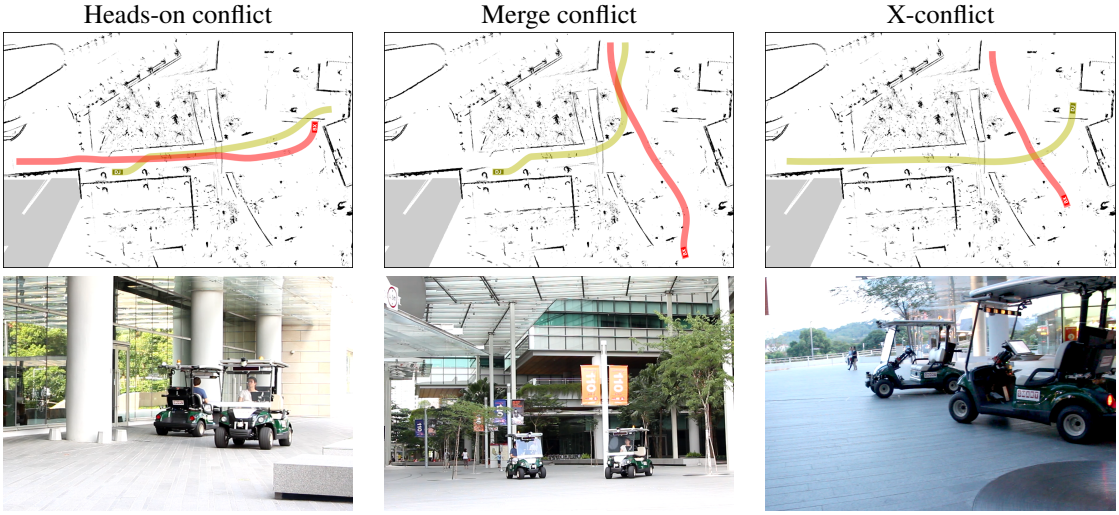
(a) Superconflict scenario – UAV missions



(b) Superconflict scenario – traces of the UAVs

For a video, visit: `https://youtu.be/Y4ZEErSyCF8`

Figure 9.4.1: Multi-UAV Testbed: Super conflict scenario.



For a video, visit: `https://youtu.be/GsFr9x29HDE`

Figure 9.4.2: AMoD System: Three selected scenarios demonstrating the behavior of CLAD-PP. The top row of figures shows the setup of each scenario, i.e., the desired path of the first robot named “DJ” (in red) and the desired path of the second robot named “BX” (in yellow). The bottom row shows the golf carts executing the coordinated trajectories.

Chapter 10

Conclusion

Safety and reliability of future multi-robot systems hinge upon the availability of *guaranteed* and *computationally tractable* method for motion coordinating among individual robots. The two existing popular approaches for multi-robot coordination are reactive methods and deliberative methods. The reactive collision-avoidance techniques are computationally efficient, however, they cannot guarantee that the robots reach their goal, i.e., they are susceptible to deadlocks. Deliberative approaches differ from the reactive approach by computing coordinated trajectories for all the robots prior execution. When the robots follow those trajectories precisely they are guaranteed to both avoid collisions and reach their goals. However, even a simple problem of disc-coordination among polygonal obstacles is known to be computationally intractable (NP-hard) and all known complete techniques have worst-case time complexity that is exponential in the number of coordinated robots, which makes them impractical for deployment in large multi-robot teams. Moreover, the coupled approaches are based on the centralized search in the joint state space of all the robots and as such are difficult to implement in a decentralized system.

Therefore, the main objective of this thesis was to study whether it is possible to design a *practically applicable* method that would guarantee the ability to resolve all conflicts between the planned trajectories of robots in the system. More specifically, we were interested in the existence of an algorithm for trajectory coordination in a multi-robot team that is guaranteed, computationally tractable, general, provide solutions of acceptable quality, and suit decentralized systems (i.e., the requirements from Section 1.1).

The central result of this thesis is an observation that in appropriately structured environments deadlock-free and computationally tractable collision avoidance is, in fact, possible to achieve. Consequently, we propose practical, yet guaranteed, centralized and decentralized algorithms for collision avoidance in multi-robot systems. The specific contributions of this thesis are the following:

1. A survey of the state of the art in single-robot and multi-robot motion planning.

2. Formalization of the concept of a well-formed infrastructure that is used to characterize a practically-relevant tractable fragment of multi-robot trajectory coordination problem and consequently enables the design of tractable guaranteed coordination algorithms for this fragment. We show that most man-made environments are structured as a well-formed infrastructure and a multi-robot system can be easily designed to satisfy the required property.

3. Tractable centralized and decentralized algorithms for multi-robot trajectory coordination with guaranteed performance in well-formed infrastructures:

   (a) Revised version of Prioritized Planning Scheme (**RPP**): An adaptation of classical prioritized planning scheme guaranteed to provide a solution for any trajectory coordination problem in well-formed infrastructures. The algorithm comes with guarantees, but remains as

scalable as classical prioritized planning, i.e., it can compute coordinated motions for tens of robots in the order of seconds.

(b) Asynchronous Decentralized version of both classical Prioritized Planning (**ADPP**) and revised prioritized planning (**ADRPP**): A decentralized version of both classical and revised prioritized planning. The algorithms are shown to terminate and ADRPP is guaranteed to provide a solution in well-formed infrastructures in the same way as its centralized counterpart. Moreover, both algorithms are able to better exploit distributed computational resources in the multi-robot team and provide a coordinated solution up to 2x-faster than the existing synchronized decentralized method.

(c) Continuous Best-Response Approach (**COBRA**): An adaptation of prioritized planning scheme to environments, where robots need to be coordinated incrementally every time a robot is assigned a task. We have shown that the algorithm is guaranteed to provide a collision-free trajectory for each possible relocation task between endpoints of a well-formed infrastructure in quadratic time in number of robots. Further, our experiments show that in large multi-robot teams the method is both more reliable and more efficient than a state-of-the-art reactive collision avoidance method.

(d) K-step Penalty Method (**kPM**): A generalization of prioritized planning that can be used as a heuristic capable of finding solutions with significantly higher quality than prioritized planning. Our experimental analysis shows that the method generates near-optimal solutions and offers better scalability in the number of robots than a state-of-the-art coupled optimal technique.

4. Robust Multi-Robot Tracking (**RMTRACK**): A control law for controlling the advancement of individual robots in the presence of exogenous delaying disturbances, such as humans stepping in the way of robots. Naive strategies for handling such disturbances are either prone to deadlocks or extremely inefficient. We show that the proposed technique provably avoids deadlocks while remaining efficient. In our experiments, we found that the proposed method scales significantly better than the baseline deadlock-free approach and furthermore it handles high-intensity delaying disturbances more reliably and more efficiently than an existing state-of-the-art reactive collision-avoidance method.

The above results were published in major robotics/automation journals and top-tier robotics/A.I. conferences. Moreover, the versatility of the proposed techniques was demonstrated by deploying asynchronous decentralized prioritized planning as a collision avoidance mechanism in two different real-world multi-robot systems: a multi-UAVs system with fixed-wing UAVs and a small automated mobility-on-demand system making use of self-driving golf carts. The source code of the above algorithms and benchmark instances used during for experimental analysis were made freely available through Github.

Future work

The research results presented in this thesis open many opportunities for follow-up work. Firstly, we proved the completeness of RPP, AD-RPP and COBRA algorithms in well-formed infrastructures for teams consisting of holonomic robots. Although our argument can be intuitively extended towards teams of non-holonomic robots, e.g., cars or airplanes, it would be interesting to validate the generalized completeness property rigorously. Secondly, we used the concept of well-formed infrastructure to characterize a class of instances of multi-robot trajectory coordination problem that can be solved efficiently in polynomial time. This class is clearly not a maximal one and thus, one can study how can be this formulation altered to cover a larger subset of all polynomially solvable instances. The third option is the study of the possibilities for parallelization of COBRA algorithm, for example by exploiting the ideas from AD-RPP method. The fourth possible venue for future work is further theoretical analysis of kPM algorithm. In particular, it would be useful to gain a better understanding of when can be the

given algorithm expected to converge to an optimal solution. Finally, it would be worthwhile investigating how can be the ideas behind RMTRACK method generalized to provide control freedom to handle unexpected disturbance not only in the time dimension but also in the two (or three) spatial dimensions.

Appendix A

Publications

Related Publications

This section lists the author's publications related to the topic of this thesis. The number in parentheses shows the contribution of the author of this thesis.

Articles in journals, with IF:

1. **M. Čáp**, P. Novák, A. Kleiner, and M. Selecký, "Prioritized planning algorithms for trajectory coordination of multiple mobile robots," *IEEE Transactions on Automation Science and Engineering*, vol. 12, iss. 3, pp. 835-849, 2015. **IF: 2.69 (70 %)**

2. A. Komenda, J. Vokřínek, **M. Čáp**, M. Pěchouček. "Developing Multiagent Algorithms for Tactical Missions Using Simulation," *IEEE Intelligent Systems,* vol. 28, iss. 1, pp. 42–49. 2013. **IF: 3.53 (10 %)**

3. M. Jakob, M. Pěchouček, **M. Čáp**, P. Novák, and O. Vaněk, "Mixed-reality testbeds for incremental development of HART applications," *IEEE Intelligent Systems*, vol. 27, iss. 2, pp. 19-25, 2012. **IF: 3.53 (20 %)**

Articles in journals:

1. B. Paden*, **M. Čáp***, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, iss. 1, pp. 33-55, 2016. **(30 %)**

   The above article was published in the first issue of a newly established IEEE journal and due its short existence it has not been assigned an impact factor yet.

In proceedings, indexed by ISI:

1. **M. Čáp***, J. Gregoire*, and E. Frazzoli, "Provably safe and deadlock-free execution of multi-robot plans under delaying disturbances," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2016)*, 2016. **(45 %)**

2. **M. Čáp**, P. Novák, M. Selecký, J. Faigl, and J. Vokřínek, "Asynchronous decentralized prioritized planning for coordination in multi-robot system," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*, 2013. **(50 %)**

In proceedings:

1. **M. Čáp**, P. Novák, and A. Kleiner, "Finding near-optimal solutions in multi-robot trajectory planning," in *Principles of Multi-robot Systems Workshop at Robotics: Science and Systems (RSS) 2015*, Rome, Italy, July 13-17, 2015., 2015. **(50 %)**

2. **M. Čáp**, J. Vokřínek, and A. Kleiner, "Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures," in *Proceedings of the Twenty-fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, Jerusalem, Israel, June 7-11, 2015. **(85 %)**

3. P. Janovský*, **M. Čáp***, and J. Vokřínek, "Finding coordinated paths for multiple holonomic robots in 2d polygonal environment," in *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014)*, 2014. **(40 %)**

4. **M. Čáp**, P. Novák, J. Vokřínek, and M. Pěchouček, "Multi-agent RRT*: sampling-based cooperative pathfinding (extended abstract)," in *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, 2013. **(60 %)**

5. M. Selecký, M. Štolba, T. Meiser, **M. Čáp**, A. Komenda, M. Rollo, J. Vokřínek, and M. Pěchouček, "Deployment of multi-agent algorithms for tactical operations on uav hardware (demonstration)," in *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, 2013. **(12.5 %)**

* Equal contribution

Unrelated Publications

This section lists the author's publications unrelated to the topic of this thesis.

Book chapters:

1. P. Novák, A. Komenda, **M. Čáp**, J. Vokřínek, and M. Pěchouček, "Simulated multi-robot tactical missions in urban warfare," in *Multiagent Systems and Applications*, Springer, 2012, pp. 147-183. **(20 %)**

In proceedings:

1. P. Novák, A. Komenda, V. Lisý, B. Bošanský, **M. Čáp**, and M. Pěchouček, "Tactical operations of multi-robot teams in urban warfare (demonstration)," in *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Richland, SC, 2012, pp. 1473-1474. **(17 %)**

2. A. Komenda, **M. Čáp**, and M. Pěchouček, "Multi-agent simulation approach to development of applications for decentralized tactical missions," in *Proceedings of Knowledge Systems for Coalition Operations (KSCO-12)*, 2012. **(25 %)**

3. **M. Čáp**, J. Vokřínek, and A. Komenda, "Communication- and computation- bounded agents in multi-agent simulations," in *Holonic and Multi-agent Systems for Manufacturing (HOLOMAS 2011)*, 2011. **(34 %)**

4. **M. Čáp**, M. Dastani, and M. Harbers, "Belief/goal sharing BDI modules (extended abstract)," in *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, 2011. **(70 %)**

5. **M. Čáp**, A. Heuvelink, K. van den Bosch, and W. van Doesburg, "Using agent technology to build a real-world training application," in Proceedings of the Workshop on Agents for Games and Simulation (AGS 2010): held at the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), 2010. **(55 %)**

Citations

Below we list all publications that received at least three citation (*excluding auto-citations*). The citation count (also excluding auto-citations) for each publication was obtained from Google Scholar database on November 22, 2016.

1. **M. Čáp**, P. Novák, A. Kleiner, and M. Selecký, "Prioritized planning algorithms for trajectory coordination of multiple mobile robots," *IEEE Transactions on Automation Science and Engineering*, vol. 12, iss. 3, pp. 835-849, 2015. **(8 citations)**

2. A. Komenda, J. Vokřínek, **M. Čáp**, M. Pěchouček. "Developing Multiagent Algorithms for Tactical Missions Using Simulation". *IEEE Intelligent Systems.* vol 28, iss. 1, pages 42–49. 2013. **(8 citations)**

3. M. Jakob, M. Pěchouček, **M. Čáp**, P. Novák, and O. Vaněk, "Mixed-reality testbeds for incremental development of HART applications," *IEEE Intelligent Systems*, vol. 27, iss. 2, pp. 19-25, 2012. **(7 citations)**

4. **M. Čáp**, P. Novák, J. Vokřínek, and M. Pěchouček, "Multi-agent RRT*: sampling-based cooperative pathfinding (extended abstract)," in *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, 2013. **(6 citations)**

5. **M. Čáp**, P. Novák, M. Selecký, J. Faigl, and J. Vokřínek, "Asynchronous decentralized prioritized planning for coordination in multi-robot system," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*, 2013. **(4 citations)**

6. **M. Čáp**, M. Dastani, and M. Harbers, "Belief/goal sharing BDI modules (extended abstract)," in *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, 2011. **(3 citations)**

Awards

The author of this thesis received two major awards:

- Joseph Fourier price for dissertation research in computer sciences, special price of Jury
- Fulbright scholarship

Bibliography

E. A. Akkoyunlu, K. Ekanandham, and R. V. Huber. Some constraints and tradeoffs in the design of network communications. In *SOSP*, pages 67–74, 1975. URL `http://dblp.uni-trier.de/db/conf/sosp/sosp75.html#AkkoyunluEH75`.

Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul Beardsley, and Roland Siegwart. *Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots*, pages 203–216. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-32723-0.

Stentz Anthony. The focussed D* algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659, 1995.

Jonathan Backer and David Kirkpatrick. Finding curvature-constrained paths that avoid polygonal obstacles. In *Proceedings of the twenty-third annual symposium on Computational geometry*, pages 66–73. ACM, 2007.

Daman Bareiss and Jur van den Berg. Generalized reciprocal collision avoidance. *The International Journal of Robotics Research*, 34(12):1501–1514, 2015.

M. Bennewitz, W. Burgard, and S. Thrun. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and Autonomous Systems*, 41(2): 89–99, 2002.

Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008. ISBN 3540779736, 9783540779735.

Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific, 1999.

John T. Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 21:193–207, 1998.

John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*, volume 19. Siam, 2010.

Amit Bhatia and Emilio Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In Rajeev Alur and George J. Pappas, editors, *Hybrid Systems: Computation and Control: 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004. Proceedings*, pages 142–156. Springer Berlin Heidelberg, 2004.

Subhrajit Bhattacharya, Vijay Kumar, and Maxim Likhachev. Distributed optimization with pairwise constraints and its application to multi-robot path planning. In *Robotics: Science and Systems*, 2010.

F. Boyer and F. Lamiraux. Trajectory deformation applied to kinodynamic motion planning for a realistic car model. In *International Conference on Robotics and Automation*, pages 487–492. IEEE, 2006.

J.F. Canny. *Complexity of robot motion planning*. MIT press, 1988.

John Canny and John Reif. New lower bound techniques for robot motion planning problems. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 49–60, Oct 1987. doi: 10.1109/SFCS.1987.42.

Michal Čáp, Peter Novák, Martin Selecký, Martin Faigl, and Jiří Vokřínek. Asynchronous decentralized prioritized planning for coordination in multi-robot system. In *Intelligent Robots and Systems (IROS), 2013*, 2013.

Michal Čáp, Peter Novák, and Alexander Kleiner. Finding near-optimal solutions in multi-robot trajectory planning. In *Principles of Multi-Robot Systems Workshop at Robotics: Science and Systems (RSS) 2015, Rome, Italy, July 13-17, 2015.*, 2015a.

Michal Čáp, Peter Novák, Alexander Kleiner, and Martin Selecký. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE Transactions on Automation Science and Engineering*, 12(3):835–849, 2015b. doi: 10.1109/TASE.2015.2445780. URL `http://dx.doi.org/10.1109/TASE.2015.2445780`.

Michal Čáp, Jiří Vokřínek, and Alexander Kleiner. Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, pages 324–332, 2015c.

Michal Čáp, Jean Gregoire, and Emilio Frazzoli. Provably safe and deadlock-free execution of multi-robot plans under delaying disturbances. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2016)*, 2016. URL `http://arxiv.org/abs/1603.08582`.

Bernard Chazelle. Approximation and decomposition of shapes. *Advances in Robotics*, 1:145–185, 1987.

Yu Fan Chen, Mark Cutler, and Jonathan P. How. Decoupled multiagent path planning via incremental sequential convex programming. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pages 5954–5961, 2015. doi: 10.1109/ICRA.2015.7140034. URL `http://dx.doi.org/10.1109/ICRA.2015.7140034`.

Z.J. Chong, B. Qin, T. Bandyopadhyay, T. Wongpiromsarn, B. Rebsamen, P. Dai, S. Kim, M.H. Ang, D. Hsu, D. Rus, and E. Frazzoli. Autonomy for mobility on demand. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4235–4236, Oct 2012. doi: 10.1109/IROS.2012.6386287.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

Kenny Daniel, Alex Nash, Sven Koenig, and Ariel Felner. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, pages 533–579, 2010.

Alex Davies. I rode 500 miles in a self-driving car and saw the future. it's delightfully dull. Wired Magazine. `http://www.wired.com/2015/01/rode-500-miles-self-driving-car-saw-future-boring/`, 2015.

Boris de Wilde, Adriaan ter Mors, and Cees Witteveen. Push and rotate: a complete multi-agent pathfinding algorithm. *Journal of Artificial Intelligence Research*, 51:443–492, 2014.

Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1: 269–271, 1959.

Jillian D'Onfro. Meet the startup that's using drones to change the world. Business Insider. http://www.businessinsider.com/matternet-uav-delivery-drones-2014-11#ixzz3UZF6yfWq, 2014.

L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79:497–516, 1957.

Michael Erdmann and Tomas Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2:1419–1424, 1987.

FAA. Fact sheet - nextgen. Federal Aviation Administration website. https://www.faa.gov/news/fact_sheets/news_story.cfm?newsId=19375, 2015.

Dave Ferguson and Anthony Stentz. Using interpolation to improve path planning: The field d* algorithm. *Journal of Field Robotics*, 23:79–101, 2006.

Cornelia Ferner, Glenn Wagner, and Howie Choset. ODrM* optimal multirobot path planning in low dimensional search spaces. In *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*, pages 3854–3859, 2013.

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

Steven Fortune and Gordon Wilfong. Planning constrained motion. *Annals of Mathematics and Artificial Intelligence*, 3:21–82, 1991.

Thierry Fraichard. Trajectory planning in a dynamic workspace: a 'state-time space' approach. *Advanced Robotics*, 13:75–94, 1998.

Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34:596–615, 1987.

Kikuo Fujimura. Time-minimum routes in time-dependent networks. *IEEE T. Robotics and Automation*, 11(3):343–351, 1995.

Sukumar Ghosh. *Distributed systems: an algorithmic approach*. CRC press, 2010.

Elena Glassman and Russ Tedrake. A quadratic regulator-based heuristic for rapidly exploring state space. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 5021–5028. IEEE, 2010.

GTAI. Industrie 4.0: Smart manufacturing for the future. German Trade and Invest Website. http://www.gtai.de/GTAI/Content/EN/Invest/_SharedDocs/Downloads/GTAI/Brochures/Industries/industrie4.0-smart-manufacturing-for-the-future-en.pdf, 2014.

Stephen. J. Guy, Jatin Chhugani, Changkyu Kim, Nadathur Satish, Ming Lin, Dinesh Manocha, and Pradeep Dubey. Clearpath: Highly parallel collision avoidance for multi-agent simulation. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 177–187, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-610-6.

Eric A Hansen and Rong Zhou. Anytime heuristic search. *J. Artif. Intell. Res.(JAIR)*, 28:267–297, 2007.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107, 1968.

J.E. Hopcroft, J.T. Schwartz, and M. Sharir. 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, December 1984.

Andrew Howard and Nicholas Roy. The robotics data set repository (radish) website. http://radish.sourceforge.net/, 2003.

David Hsu, Jean-Claude Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In *International Conference on Robotics and Automation*, volume 3, pages 2719–2726. IEEE, 1997.

David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255, 2002.

Paul Jacobs and John Canny. Planning smooth paths for mobile robots. In *Nonholonomic Motion Planning*, pages 271–342. Springer, 1993.

Michal Jakob, Michal Pěchouček, Peter Novák, Michal Čáp, and Ondra Vaněk. Towards incremental development of human-agent-robot applications using mixed-reality testbeds. *IEEE Intelligent Systems*, 27(2):19–25, 2012.

Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research*, 2015.

Jeong H. Jeon, Raghvendra V Cowlagi, Steven C Peters, Sertac Karaman, Emilio Frazzoli, Panagiotis Tsiotras, and Karl Iagnemma. Optimal motion planning with the half-car dynamical model for autonomous high-speed driving. In *American Control Conference*, pages 188–193. IEEE, 2013.

K Kant and S W Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *Int. J. Rob. Res.*, 5(3):72–89, September 1986. ISSN 0278-3649. doi: 10.1177/027836498600500304. URL `http://dx.doi.org/10.1177/027836498600500304`.

Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, June 2011.

Sertac Karaman and Emilio Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *International Conference on Robotics and Automation*, pages 5041–5047. IEEE, 2013.

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

Lydia E Kavraki, Mihail N Kolountzakis, and J-C Latombe. Analysis of probabilistic roadmaps for path planning. *Transactions on Robotics and Automation*, 14:166–171, 1998.

Sven Koenig and Maxim Likhachev. Fast replanning for navigation in unknown terrain. *Transactions on Robotics*, 21:354–363, 2005.

Daniel Kornhauser, Gary Miller, and Paul Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *25th Annual Symposium on Foundations of Computer Science. 1984*, pages 241–250. IEEE, 1984.

Tobias Kunz and Mike Stilman. Kinodynamic RRTs with fixed time step and best-input extension are not probabilistically complete. In *Algorithmic Foundations of Robotics XI*, pages 233–244. Springer, 2015.

Steven M La Valle. Rapidly-exploring random trees a new tool for path planning. Technical report, Computer Science Dept., Iowa State University, 1998.

F. Lamiraux, E. Ferre, and E. Vallee. Kinodynamic motion planning: Connecting exploration trees using trajectory optimization methods. In *International Conference on Robotics and Automation*, volume 4, pages 3987–3992. IEEE, 2004.

Frederic Lardinois and Alex Wilhelm. Ford has big plans for autonomous cars and the future of driving. TechCrunch website. `http://techcrunch.com/2015/01/06/ford-has-big-plans-for-autonomous-cars-and-the-future-of-driving/`, 2015.

Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

Steven M. LaValle and James J. Kuffner. Randomized kinodynamic planning. *I. J. Robotic Res.*, 20(5): 378–400, 2001.

Steven M. LaValle, Michael S. Branicky, and Stephen R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *I. J. Robotic Res.*, 23(7-8):673–692, 2004.

Sylvain Lazard, John Reif, and Hongyan Wang. The complexity of the two dimensional curvature-constrained shortest-path problem. In *Proceedings of the Third International Workshop on the Algorithmic Foundations of Robotics*, pages 49–57, 1998.

Jed Lengyel, Mark Reichert, Bruce R Donald, and Donald P Greenberg. *Real-time robot motion planning using rasterizing computer graphics hardware*, volume 24. ACM, 1990.

Jacques Leslie. Free flight. Wired Magazine. `http://archive.wired.com/wired/archive/4.04/es.faa.html`, 1996.

Yanbo Li, Zakary Littlefield, and Kostas E Bekris. Sparse methods for efficient asymptotically optimal kinodynamic planning. In *Algorithmic Foundations of Robotics XI*, pages 263–282. Springer, 2015.

Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems*, page None, 2003.

Maxim Likhachev, David I Ferguson, Geoffrey J Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic A*: An anytime, replanning algorithm. In *ICAPS*, pages 262–271, 2005.

Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

Vladimir J. Lumelsky and KR Harinarayan. Decentralized motion planning for multiple mobile robots: The cocktail party model. In *Robot colonies*, pages 121–135. Springer, 1997.

Ryan Luna and Kostas E. Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In *International Joint Conference on Artificial Intelligence 2011*, pages 294–300, 2011.

Eric Mack. Elon musk: Don't fall asleep at the wheel for another 5 years. CNET Roadshow Website. `http://www.cnet.com/news/elon-musk-sees-autonomous-cars-ready-sooner-than-previously-thought/`, 2014.

Friedemann Mattern. Algorithms for distributed termination detection. *Distributed computing*, 2(3): 161–175, 1987.

Daniel Mellinger, Alex Kushleyev, and Vijay Kumar. Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 477–483. IEEE, 2012.

Venkatraman Narayanan, Mike Phillips, and Maxim Likhachev. Anytime safe interval path planning for dynamic environments. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4708–4715. IEEE, 2012.

Alex Nash, Sven Koenig, and Craig Tovey. Lazy theta*: Any-angle path planning and path length analysis in 3d. In *Third Annual Symposium on Combinatorial Search*, 2010.

Nils J Nilsson. A mobile automaton: An application of artificial intelligence techniques. Technical report, DTIC Document, 1969.

Raz Nissim and Ronen Brafman. Distributed heuristic forward search for multi-agent planning. *J. Artif. Int. Res.*, 51(1):293–332, September 2014. ISSN 1076-9757. URL `http://dl.acm.org/citation.cfm?id=2750423.2750431`.

Patrick A O'Donnell and T Lozano-Periz. Deadlock-free and collision-free coordination of two robot manipulators. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 484–489. IEEE, 1989.

Colm Ó'Dúnlaing and Chee K Yap. A "retraction" method for planning the motion of a disc. *Journal of Algorithms*, 6:104–111, 1985.

Michael Otte and Emilio Frazzoli. RRT-X: Real-time motion planning/replanning for environments with unpredictable obstacles. In *International Workshop on the Algorithmic Foundations of Robotics*, 2014.

Mark H Overmars and Emo Welzl. New methods for computing visibility graphs. In *Proceedings of the fourth annual symposium on Computational geometry*, pages 164–171. ACM, 1988.

Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, March 2016. ISSN 2379-8858. doi: 10.1109/TIV.2016.2578706.

Alejandro Perez, Robert Platt Jr, George Konidaris, Leslie Kaelbling, and Tomas Lozano-Perez. LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *International Conference on Robotics and Automation*, pages 2537–2542. IEEE, 2012.

Stephane Petti and Thierry Fraichard. Safe motion planning in dynamic environments. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2210–2215. IEEE, 2005.

Mike Phillips and Maxim Likhachev. Sipp: Safe interval path planning for dynamic environments. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5628–5635. IEEE, 2011.

Mihail Pivtoraiko, Ross A. Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009. ISSN 1556-4967. doi: 10.1002/rob.20285. URL `http://dx.doi.org/10.1002/rob.20285`.

Ira Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, pages 219–236, 1970.

E. Polak. An historical survey of computational methods in optimal control. *SIAM Review*, 15:553–584, 1973.

Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. CRC Press, 1987.

Daniel Ratner and Manfred K Warmuth. Finding a shortest solution for the n× n extension of the 15-puzzle is intractable. In *AAAI*, pages 168–172, 1986.

James Reeds and Lawrence Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific journal of mathematics*, 145:367–393, 1990.

John Reif and Micha Sharir. Motion planning in the presence of moving obstacles. *J. ACM*, 41(4): 764–790, July 1994. ISSN 0004-5411. doi: 10.1145/179812.179911. URL `http://doi.acm.org/10.1145/179812.179911`.

John H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, SFCS '79, pages 421–427, Washington, DC, USA, 1979. IEEE Computer Society.

Gabriele Röger and Malte Helmert. Non-optimal multi-agent pathfinding is solved (since 1984). In *Proceedings of the Fifth Annual Symposium on Combinatorial Search (SoCS 2012)*, pages 173–174, 2012.

Rebecca Rosen. Google's self-driving cars: 300,000 miles logged, not a single accident under computer control. The Atlantic. `http://www.theatlantic.com/technology/archive/2012/08/googles-self-driving-cars-300-000-miles-logged-not-a-single-accident-under-computer-control/260926/`, 2012.

Michael Ross and Mark Karpenko. A review of pseudospectral optimal control: from theory to flight. *Annual Reviews in Control*, 36(2):182–197, 2012.

Alessandro Rucco, Giuseppe Notarstefano, and John Hauser. Computing minimum lap-time trajectories for a single-track car with load transfer. In *Conference on Decision and Control*, pages 6321–6326. IEEE, 2012.

Edward Schmerling, Lucas Janson, and Marco Pavone. Optimal sampling-based motion planning under differential constraints: the driftless case. *International Conference on Robotics and Automation*, 2015.

Tom Schouwenaars, Bart De Moor, Eric Feron, and Jonathan How. Mixed integer programming for multi-vehicle path planning. In *European control conference*, volume 1, pages 2603–2608, 2001.

Jacob T Schwartz and Micha Sharir. On the "piano movers" problem. II. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4(3): 298 – 351, 1983a. ISSN 0196-8858. doi: http://dx.doi.org/10.1016/0196-8858(83)90014-3. URL `http://www.sciencedirect.com/science/article/pii/0196885883900143`.

Jacob T Schwartz and Micha Sharir. On the piano movers' problem: III. coordinating the motion of several independent bodies: the special case of circular bodies moving amidst polygonal barriers. *The International Journal of Robotics Research*, 2(3):46–75, 1983b.

Martin Selecký, Michal Štolba, Tomáš Meiser, Michal Čáp, Antonín Komenda, Milan Rollo, Jiří Vokřínek, and Michal Pěchouček. Deployment of multi-agent algorithms for tactical operations on uav hardware. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, pages 1407–1408. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.*, 219:40–66, 2015. doi: 10.1016/j.artint.2014.11.006. URL http://dx.doi.org/10.1016/j.artint.2014.11.006.

Kiril Solovey and Dan Halperin. On the hardness of unlabeled multi-robot motion planning. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015.

Paul G. Spirakis and Chee-Keng Yap. Strong NP-hardness of moving many discs. *Inf. Process. Lett.*, 19(1):55–59, 1984.

Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2010. URL http://dblp.uni-trier.de/db/conf/aaai/aaai2010.html#Standley10.

Trevor Scott Standley and Richard E. Korf. Complete algorithms for cooperative pathfinding problems. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 668–673. IJCAI/AAAI, 2011.

Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *International Conference Robotics and Automation*, pages 3310–3317. IEEE, 1994.

James A. Storer and John H. Reif. Shortest paths in the plane with polygonal obstacles. *J. ACM*, 41(5):982–1012, September 1994. ISSN 0004-5411. doi: 10.1145/185675.185795. URL http://doi.acm.org/10.1145/185675.185795.

Pavel Surynek. A novel approach to path planning for multiple robots in bi-connected graphs. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, ICRA'09, pages 928–934, Piscataway, NJ, USA, 2009a. IEEE Press. ISBN 978-1-4244-2788-8. URL http://dl.acm.org/citation.cfm?id=1703435.1703586.

Pavel Surynek. On pebble motion on graphs and abstract multi-robot path planning. In *Proceedings of the ICAPS 2009 Workshop on Generalized Planning*, pages 2–9, 2009b.

Osamu Takahashi and Robert J Schilling. Motion planning in a plane using generalized voronoi diagrams. *Transactions on Robotics and Automation*, 5:143–150, 1989.

Jur van den Berg and Mark Overmars. Prioritized motion planning for multiple robots. In *IEEE/RSJ International Conference on Intelligent Robots*, pages 430–435, 2005a.

Jur van den Berg and Mark Overmars. Roadmap-based motion planning in dynamic environments. *Robotics, IEEE Transactions on*, 21(5):885–897, 2005b.

Jur van den Berg and Mark Overmars. Kinodynamic motion planning on roadmaps in dynamic environments. In *IEEE/RSJ International Conference on Intelligent Robots*, pages 4253–4258. IEEE, 2007.

Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1928–1935. IEEE, 2008.

Jur van den Berg, Stephen Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. *Robotics Research*, pages 3–19, 2011.

Jakub Vašek. Analýza použitelnosti pebble-motion algoritmů pro koordinaci trajektorií v multi-robotickém týmu. Bc. Thesis, CTU in Prague, 2015.

Prasanna Velagapudi, Katia P. Sycara, and Paul Scerri. Decentralized prioritized planning in large multirobot teams. In *IEEE/RSJ International Conference on Intelligent Robots*, pages 4603–4609. IEEE, 2010. ISBN 978-1-4244-6674-0. URL http://dblp.uni-trier.de/db/conf/iros/iros2010.html#VelagapudiSS10.

Glenn Wagner and Howie Choset. M*: A complete multirobot path planning algorithm with performance bounds. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2011,*, pages 3260–3267, 2011.

Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1 – 24, 2015. ISSN 0004-3702.

Hongyan Wang and Pankaj K Agarwal. Approximation algorithms for curvature-constrained shortest paths. In *SODA*, volume 96, pages 409–418, 1996.

David J Webb and Jur van den Berg. Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In *International Conference on Robotics and Automation*, pages 5054–5061. IEEE, 2013.

Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In *Proceedings of the 19th National Conference on Innovative Applications of Artificial Intelligence - Volume 2*, IAAI'07, pages 1752–1759. AAAI Press, 2007. ISBN 978-1-57735-323-2. URL http://dl.acm.org/citation.cfm?id=1620113.1620125.

Peter Yap, Neil Burch, Robert C Holte, and Jonathan Schaeffer. Block a*: Database-driven search with applications in any-angle path-planning. In *AAAI*, 2011.

J. Yu. Intractability of optimal multi-robot path planning on planar graphs. *IEEE Robotics and Automation Letters*, 1(1):33–40, 2016.

J. Yu and S. M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *The Twenty-Seventh AAAI Conference on Artificial Intelligence*, pages 1444–1449, 2013.

J. Yu and S. M. LaValle. Optimal Multi-Robot Path Planning on Graphs: Complete Algorithms and Effective Heuristics. *ArXiv e-prints*, July 2015.