Czech Technical University in Prague
Faculty of Electrical Engineering

DIPLOMA THESIS

Increasing Coverage of Reactive Techniques for Collision Avoidance

Prague, 2013 **Author: Bc. Pavel Janovský**

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 18. 12. 2013

Janovský

Podpis autora práce

Acknowledgements

First and foremost I thank to my adviser Bc. Michal Čáp MSc. from Agent Technology Center, Czech Technical University, for a guidance throughout the work on this thesis. Furthermore I want to thank to my family for their support during my work on the thesis.

Abstrakt

Koordinace multiagentních cest je zásadní pro navigaci robotů ve společném prostředí. V této praci studujeme problém koordinace multiagentních cest ve spojitém prostoru s polygonálními překážkami. Tento problém je těžký, protože může vyžadovat plánování ve stavovém prostoru exponencálním k počtu agentů. Dobře známé techniky, které řeší tento problém, jsou reaktivní techniky pro vyhýbání robotů a plánování založené na vzorkování (sampling based planning). V této práci uvádíme přehled často používaného přístupu k reaktivním technikám - rychlostní překážky (velocity obstacle). Také studujeme plánování založené na vzorkování, zaměřujeme se na dobře známý algoritmus RRT*.

Nejmodernější reaktivní technika pro vyhýbání robotů - ORCA, i RRT* algoritmus mají omezené pokrytí prostoru instancí problému. Navrhujeme nový algoritmus - ORCA-RRT*, který kombinuje obě zmíněné techniky. Zjistili jsme, že tento přístup těží z obou svých částí, díky čemuž poskytuje lepší pokrytí prostoru instancí problému spolu s vyšší kvalitou poskytnutých řešení.

Srovnáváme výkon našeho ORCA-RRT* algoritmu s reaktivní technikou a s plánovacím algoritmem RRT*. Experimentálně ukazujeme, že a) ORCA-RRT* je schopen překonat lokální minima, která se vyskytují v mnoha instancích problému s hustším prostředím a způsobují zásadní pokles výkonu reaktivní techniky pro vyhýbání a b) ORCA-RRT* řeší mnoho instancí s vysokým počtem agentů, které jiné algoritmy založené na RRT* nemohou vyřešit kvůli exponenciálnímu růstu stavového prostoru s počtem agentů.

Abstract

Multi-agent path coordination is essential for navigation of multiple robots in a common environment. In this thesis we study a problem of multi-agent path coordination in a continuous space with polygonal obstacles. This problem is challenging because it may require planning in a state space exponential to the number of agents. The well known techniques that are able to solve this problem are reactive collision avoidance and sampling based planning. In this thesis we provide an overview of a widely used approach for reactive collision avoidance - a velocity obstacle. We also study the sampling based planning approach, focusing on a well known RRT* algorithm.

A state of the art reactive collision avoidance technique - ORCA, as well as the RRT* algorithm both have limited coverage of the problem instance space. We propose a new algorithm - ORCA-RRT*, which combines both mentioned techniques. We find that this approach is able to benefit from both its parts resulting in better coverage of the problem instance space along with higher quality of the provided solutions.

We compare the performance of our ORCA-RRT* algorithm with the reactive technique as well as the RRT* planning algorithm. We experimentally show that a) ORCA-RRT* is able to overcome local minima, which occur in many dense problem instances and cause a significant decrease in the performance of the local reactive collision avoidance technique and b) ORCA-RRT* solves many instances with high number of agents, which other RRT*-based algorithms are not able to solve due to the exponential growth of the state space with the number of agents.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	Bc. Pavel J a n o v s k ý
Studijní program:	Otevřená informatika (magisterský)
Obor:	Umělá inteligence
Název tématu:	Zvýšení úspěšnosti reaktivních technik pro vyhýbání mobilních robotů

Pokyny pro vypracování:

1. Seznamte se s reaktivními technikami pro vyhýbání mobilních robotů.
2. Analyzujte nedostatky vybrané reaktivní techniky a identifikujte typické případy, kdy tato technika selhává.
3. Navrhněte rozšíření vybrané techniky zvyšující její úspěšnost na identifikovaných problematických případech.
4. Navržené řešení implementujte a experimentálně ověřte.

Seznam odborné literatury:

[1] Emmett Lalish and Kristi A. Morgansen: Distributed Reactive Collision Avoidance. Autonomous Robots 32.3 (2012) : 207-226.
[2] Jur van den Berg, et al.: Reciprocal n-Body Collision Avoidance. Robotics Research. Springer Berlin Heidelberg, 2011. 3-19.
[3] Stephen J. Guy, et al.: ClearPath: Highly Parallel Collision Avoidance for Multi-Agent Simulation. Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. ACM, 2009.

Vedoucí diplomové práce: Bc. Michal Čáp, MSc.

Platnost zadání: do konce zimního semestru 2014/2015

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická -4- L.S.

[illegible]

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

[illegible]

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 25. 6. 2013

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student:	Bc. Pavel J a n o v s k ý
Study programme:	Open Informatics
Specialisation:	Artificial Intelligence
Title of Diploma Thesis:	Increasing Coverage of Reactive Techniques for Collision Avoidance

Guidelines:

1. Review existing reactive techniques for collision avoidance.
2. Analyze weaknesses of a selected technique and identify typical scenarios, where the technique fails.
3. Design an extension of the selected technique that will increase its coverage of the identified scenarios.
4. Implement the extension and experimentally evaluate.

Bibliography/Sources:

[1] Emmett Lalish and Kristi A. Morgansen: Distributed Reactive Collision Avoidance. Autonomous Robots 32.3 (2012) : 207-226.
[2] Jur van den Berg, et al.: Reciprocal n-Body Collision Avoidance. Robotics Research. Springer Berlin Heidelberg, 2011. 3-19.
[3] Stephen J. Guy, et al.: ClearPath: Highly Parallel Collision Avoidance for Multi-Agent Simulation. Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. ACM, 2009.

Diploma Thesis Supervisor: Bc. Michal Čáp, MSc.

Valid until: the end of the winter semester of academic year 2014/2015

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
L.S.
-4-

[illegible]

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

[illegible]

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, June 25, 2013

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays we can witness an increasing usage of robots working in teams, for instance Automatic Guided Vehicle Systems - systems composed of multiple robots that operate in shared environment without a need of a human driver. According to (Miller, 2012), the industry's leading suppliers of automatic guided vehicle systems, in 2012 the market saw dollar sales of more than \$100 million, which compared to 2007 is nearly a 25% increase in sales. Given this statistic an issue of coordination between robots on various levels arises.

In this thesis we address a problem of path coordination between multiple physical agents. We assume disc-shaped physical agents with holonomic kinematics that navigate in 2-d environment with polygonal obstacles. The task is to navigate the agents through the environment in order to reach their goal positions such that the collisions between the agents are avoided. We will refer to this problem as a multi-agent path coordination problem.

The problem of finding coordinated non-conflicting paths for a group of objects in restricted 2-d environment was proved to be in PSPACE-hard (Hopcroft et al., 1984). Recently there have been accomplishments in solving a restricted problem, where the agents move on a graph. Algorithms such as BIBOX (Surynek, 2009) or Push & Rotate (de Wilde et al., 2013) solve the restricted problem in polynomial time. These algorithms assume point like agents and therefore cannot be used in systems where the size of the agents cannot be neglected. Prioritized planning (Erdmann and Lozano-Perez, 1986), where agents plan their paths on a graph in the order of their priorities, is efficient especially in uncluttered environments. It is not clear how to remove the graph constraint of prioritized planning so it could be used without discretization on continuous environments. Path coordination between multiple agents in continuous environment is usually solved with reactive collision avoidance techniques. Especially the research of velocity obstacles (Fiorini and Shiller, 1998; van den Berg et al., 2011; van den Berg and Manocha, 2008; Guy et al., 2009; Lalish and Morgansen, 2009) yields promising results for agent coordination. Even though this approach is frequently applied, there are several challenges associated with it such as its demand on fast online calculation, risk of oscillations or risk of getting stuck in local minima. We will describe these challenges as well as the extent to which they have been addressed in the previous research in Chapter 2. Optimal path for a single agent in a continuous state space can be found using some sampling-based algorithm such as a recently proposed RRT* (Karaman and Frazzoli, 2011), an anytime extension of RRT (LaValle and Kuffner, 1999). RRT* addresses the problem of single agent path finding, but can be extended to multiple agent path coordination, as proposed in (Cap et al., 2013), which solves a restricted problem, where the agents move towards their goals on a graph. While this approach is able to solve the problem for several agents moving on a graph, its success rate decreases fast for higher numbers of agents due to the exponential growth of its state space. We propose several RRT* based algorithms capable of solving the multi-agent path coordination problem in the continuous state space, i.e. without any of the above mentioned restrictions, in Chapter 2.

We propose a new algorithm - ORCA-RRT*, which builds upon the reactive collision avoidance techniques and extends them using RRT* planning. It uses planning to overcome the above mentioned risk of the reactive collision avoidance techniques - getting stuck in a local minima. We show that the ORCA-RRT* algorithm is able to benefit from both its parts to overcome the mentioned problems and therefore it increases the coverage of the problem instance space.

Our experiment results show that the coverage of the instance space by both RRT* planning and reactive collision avoidance is significantly restricted in various parts of the problem instance space. On the other hand our ORCA-RRT* algorithm is able to cover the union of the coverage of its planning and reactive parts. Further we found that ORCA-RRT* provides solutions of better quality than the reactive techniques and other RRT* based algorithms.

This thesis is organized as follows: in Chapter 2 we summarize the related work in reactive collision avoidance and multi-agent planning. Then, we define the multi-agent path coordination problem in Chapter 3. Chapter 4 exposes the proposed ORCA-RRT* algorithm, which we then analyze theoretically in Chapter 5 and experimentally in Chapter 6. Finally we conclude the thesis in Chapter 7.

Chapter 2

Related Work

In this chapter we summarize the state of the art techniques used in the field of multi-agent path coordination in continuous environments. In literature we can find two main approaches to the problem: reactive collision avoidance and sampling based planning. Both of these approaches can be used to solve the multi-agent path coordination problem, which we formally define in Chapter 3. There are several differences in these approaches. We summarize the main differences, advantages and disadvantages of reactive collision avoidance and sampling based planning approaches in Table 2.1.

In Table 2.1 we state that sampling based planning is probabilistically complete and asymptotically optimal, which was proved in (Karaman and Frazzoli, 2011). On the other hand reactive collision avoidance techniques are incomplete, because they only search the space locally. In the experimental analysis we found feasible problem instances that cannot be solved by reactive collision avoidance (see Figure 4.1). The fact that sampling based planning is not able to solve a problem for high number of agents comes from our experimental analysis and the fact that the planning state space grows exponentially to the number of agents.

In Section 2.1 we describe the frequently used reactive collision avoidance techniques. In Section 2.2 we then briefly describe the sampling based planning approach to path finding problem.

	Reactive Collision Avoidance	Sampling Based Planning
Continuous environment	Yes	Yes
Computation	Online	Offline
Complete	No	Yes - probabilistically
Optimal	No	Yes - asymptotically
Ability to solve the problem for high number of agents (> 100)	Yes	No
Ability to compensate sensors and actuators inaccuracy	Yes	No

Table 2.1: Main differences between reactive collision avoidance and planning

2.1 Reactive Techniques for Collision Avoidance

Collision avoidance is an online approach for solving multi-agent path coordination problem where multiple agents navigate through an environment applying a repeating sequence of sensing and acting in order to avoid collisions between each other and with obstacles and reach their predefined goals. It is a decoupled approach, each robot decides autonomously about its future actions. Vast majority of reactive collision avoidance techniques in the literature is based on a velocity obstacle paradigm.

2.1.1 Velocity Obstacle for navigating a single agent among dynamic obstacles

Robot path finding among dynamic obstacles has been studied for over twenty years, but mainly using offline planning techniques. A breakthrough result in reactive collision avoidance appeared in 1998, when P. Fiorini and Z. Shiller proposed a velocity obstacle concept (Fiorini and Shiller, 1998). Velocity obstacle approach was first that was able to find a path for a single agent in a dynamic environment using velocities of moving objects. The method was described as being a first order method - a method that uses velocities of objects for collision detection and avoidance. Until then the robot path finding problem was solved by zero order methods which used only position information to determine possible collisions (Fiorini and Shiller, 1998). The robot path finding problem was usually solved by time extended visibility-graph approach (Reif and Sharir, 1985). By relying on velocity information the velocity obstacle approach is able to consider robot dynamics.

The velocity obstacle is defined in (Fiorini and Shiller, 1998) as a first-order approximation of the robot's velocities that would cause a collision with a static or dynamic obstacle at some future time, within a given time horizon. The velocity obstacle for an agent A induced by a moving obstacle B is created as depicted in Figure 2.1. We first map B into the configuration space of A by extending the radius of B with the radius of A and assuming that A has a zero radius. A set of relative collision velocities - a collision cone for an agent A induced by an obstacle B is then formally defined as (Fiorini and Shiller, 1998):

$$CC_{A|B} = \{v_{A,B} | \lambda_{A,B} \cap D(p_B, r_A + r_B) \neq \emptyset\}, \tag{2.1}$$

where $v_{A,B}$ is the relative velocity of A with respect to B, $v_{A,B} = v_A - v_B$, $\lambda_{A,B}$ is a line of $v_{A,B}$, r_A, r_B are the radii of the agent A and the obstacle B respectively, p_B is the position of the obstacle B and $D(p_B, r_A + r_B)$ denotes a disc centered in p_B with radius $r_A + r_B$, which is the disc extended from B by r_A (see Figure 2.1).

Any relative velocity $v_{A,B}$ that lies in $CC_{A|B}$ will lead to a collision and we call it conflicting velocity. Since the agent has to deal with multiple moving obstacles, it is convenient to create a set of absolute conflicting velocities of the agent - a velocity obstacle. This is done by translating the collision cone by velocity v_B. Formally the velocity obstacle for an agent A induced by an obstacle B is defined as:

$$VO_{A|B} = CC_{A|B} \oplus v_B, \tag{2.2}$$

where $\oplus$ is the Minkowski sum operator [1].

The agent A creates a velocity obstacle for all obstacles in the environment. The resulting set of conflicting velocities - the velocity obstacle for the agent A, is created by a union of all computed obstacles:

$$VO_A = \bigcup_{i=1}^{m} VO_{A|B_i}, \tag{2.3}$$

where m is the number of obstacles in the environment.

Since this velocity obstacle restricts the velocity space significantly, it is convenient to introduce a time horizon parameter τ and only consider collisions that happen before

[1] Minkowski sum of two sets of vectors $\mathbf{A}$ and $\mathbf{B}$ is defined as $\mathbf{A} \oplus \mathbf{B} = \{a + b | a \in \mathbf{A}, b \in \mathbf{B}\}$

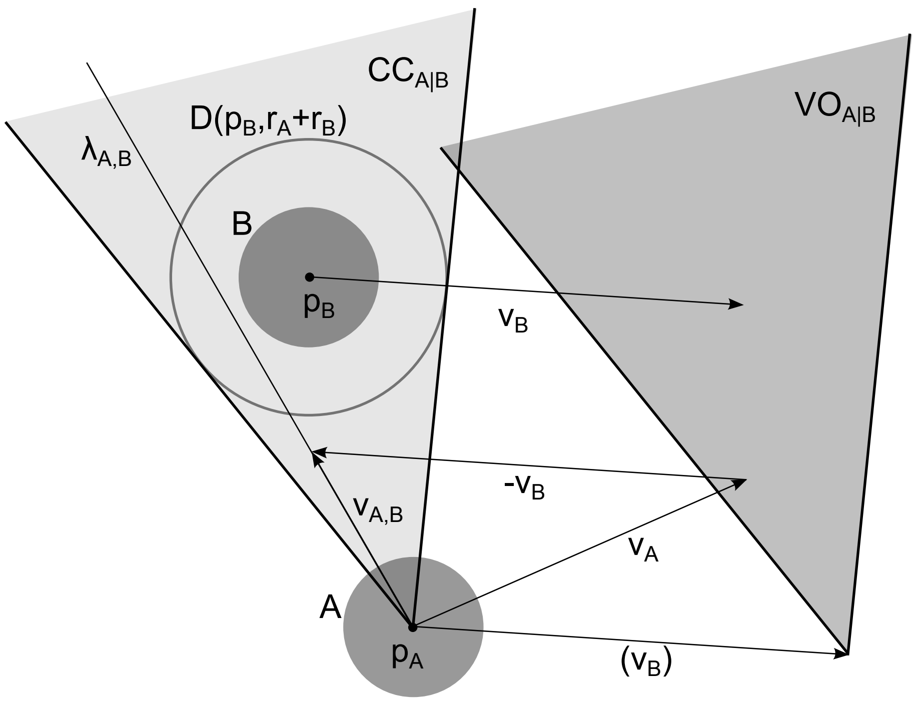


Figure 2.1: Velocity obstacle for agent A induced by obstacle B

the time τ. This adds an additional condition to the velocity obstacle definition in Equation 2.3. The specific way of formalizing such a condition differs in the various papers on velocity obstacles mentioned in this thesis, (Fiorini and Shiller, 1998) defines the velocity obstacle with respect to the time horizon as

$$VO_A^{\tau} = VO_A \setminus \{v_A | v_A \in VO_A, \|v_{A,B}\| \leq \frac{dist(A,B)}{\tau}\}, \tag{2.4}$$

where $dist(A, B)$ is the Euclidean distance between A and B.

This velocity obstacle divides the space into two parts - conflicting and not conflicting velocities. As we mentioned earlier, reactive collision avoidance is typically assumed to be an online approach, where the agents are computing the velocity obstacles and new velocities while they are navigating in the environment. Therefore, from all the non-conflicting velocities $v \notin VO_A$ one velocity vector has to be chosen in each time step. This velocity will be applied by the agent. There are several ways of choosing this new velocity based on the computed velocity obstacles, some desired velocity and possibly some other constraints which arise from the dynamic model of the robot. In (Fiorini and Shiller, 1998) the authors also present a search based approach, where the space of non-conflicting velocities is discretized and the agent plans the actions offline, before navigating through the environment. The later papers on velocity obstacles (van den

Berg and Manocha, 2008; van den Berg et al., 2011; Lalish and Morgansen, 2009) only consider the online approach.

2.1.2 Reciprocal Velocity Obstacle

So far the concept of velocity obstacles was used to navigate a single agent in an environment with static and dynamic obstacles, where the dynamic obstacles do not react in any way to movements of other obstacles. If the velocity obstacle approach from (Fiorini and Shiller, 1998) is used for multi-agent navigation, where each agent would consider all other agents as moving obstacles, an oscillation behavior may occur between the agents because they all take full responsibility for the collision avoidance. If two agents were close to a collision, they would both change their velocities. This would move their velocity obstacles and thus put their old velocities outside of the constrained velocity space. In the next step the agents would therefore again choose their old velocities, which creates an oscillation loop (van den Berg and Manocha, 2008).

An adaptation of the velocity obstacle concept that attempts to deal with the oscillation phenomenon was presented as a Reciprocal Velocity Obstacle (RVO) (van den Berg and Manocha, 2008). Reciprocal velocity obstacle is an approach similar to velocity obstacle, that further assumes that all the agents make a similar collision avoidance reasoning. Under this assumption each agent takes a partial responsibility for the collision avoidance and therefore the RVO is guaranteed to generate safe and oscillation-free motions.

The reciprocal velocity obstacle works as follows. Lets first assume that each agent will take half of the responsibility for the collision avoidance. Therefore instead of choosing a new velocity outside of the velocity obstacle, it chooses a velocity that is an average of a velocity outside of the velocity obstacle and its current velocity. This choice is equivalent to a translation of the velocity obstacle such that its apex lies in $\frac{v_A+v_B}{2}$ instead of v_B. The situation is depicted in Figure 2.2.

The reciprocal velocity obstacle for agent A induced by agent B is defined as (van den Berg and Manocha, 2008)

$$RVO_{A|B} = \{v'_A | 2v'_A - v_A \in VO_{A|B}\}. \tag{2.5}$$

The approach can be generalized in a way that allows us to assign priorities to agents according to which they would be responsible for the collision avoidance. The definition

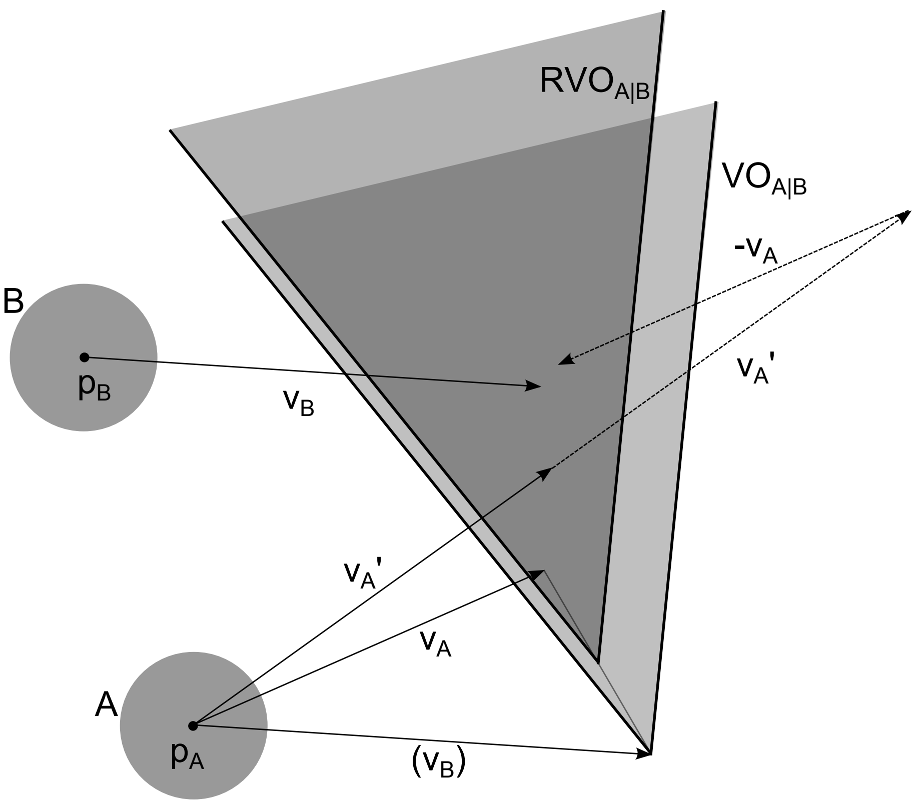


Figure 2.2: Reciprocal velocity obstacle for agent A induced by agent B

of generalized RVO is (van den Berg and Manocha, 2008)

$$RVO^{G}_{A|B} = \{v'_A | \frac{1}{\alpha_{A,B}} v'_A + (1 - \frac{1}{\alpha_{A,B}}) v_A \in VO_{A|B}\}, \tag{2.6}$$

where $\alpha_{A,B}$ is the share of effort agent A takes to avoid agent B.

2.1.3 Optimal Reciprocal Collision Avoidance

A method that further improves the RVO was published recently in (van den Berg et al., 2011). The authors propose a locally optimal approach called optimal reciprocal collision avoidance - ORCA. They differentiate from (van den Berg and Manocha, 2008) by stating that RVO guarantees collision avoidance only under specific conditions. While both agents selecting a velocity inside each other's RVO is a sufficient condition to result in a collision, the converse does not hold.

Before describing the ORCA algorithm, we note that the way the velocity obstacles are computed in ORCA is slightly different from (Fiorini and Shiller, 1998). Figure 2.3 shows how the velocity obstacle is computed, given the time threshold τ. Also note that

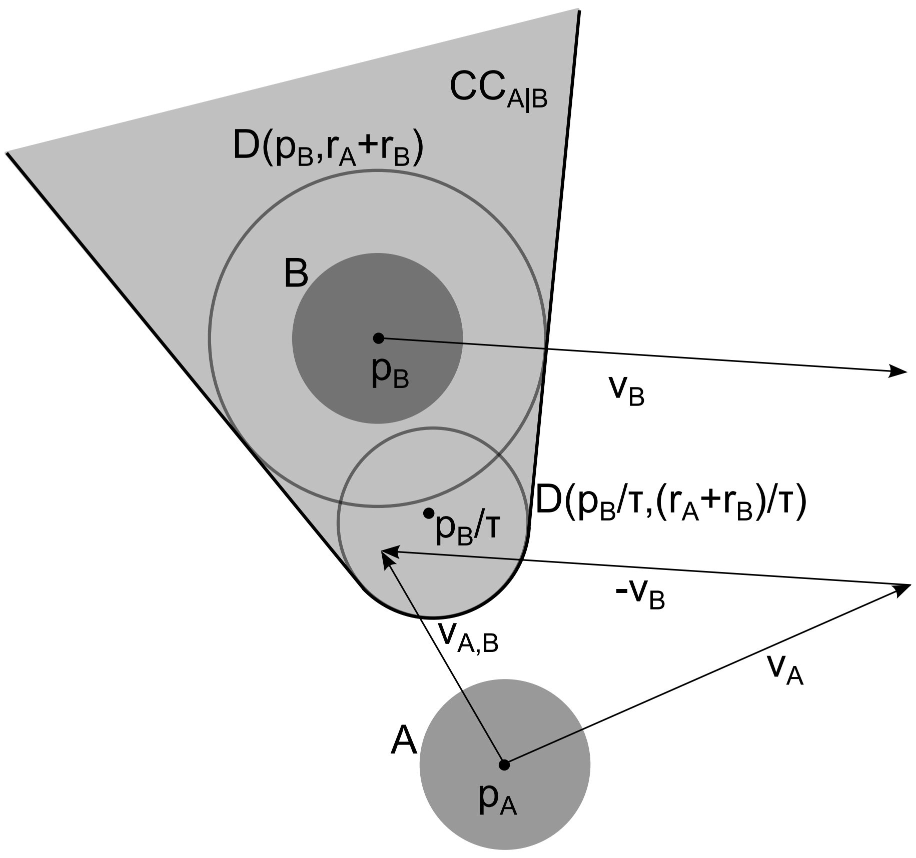


Figure 2.3: Velocity obstacle created by ORCA for agent A induced by agent B

the ORCA algorithm only computes the relative velocity obstacle, which in our notation is called a collision cone (see Figure 2.1).

While the way ORCA calculates the velocity obstacles does not differ from the method used in (Fiorini and Shiller, 1998; van den Berg and Manocha, 2008) significantly, the method for computing the agent's new velocity is different. After the agent computes a velocity obstacle, an ORCA line is created. An ORCA line is a linear velocity space constraint which divides the velocity space into two half planes, one of non-conflicting velocities, the other of conflicting velocities. Note that while the ORCA algorithm builds upon the velocity obstacle concept, it uses ORCA lines instead of the velocity obstacles to constraint the velocity space. First a vector u from the relative velocity $v_{A,B}$ to the closest point on the velocity obstacle boundary is found. Then a vector n normal to the boundary is constructed from the end point of the vector u. The ORCA line is then given by a point $v_A + \frac{1}{2}u$ and a vector perpendicular to n as the direction of the line. The process is shown in Figure 2.4. The second term in the formula - $\frac{1}{2}u$, formalizes the

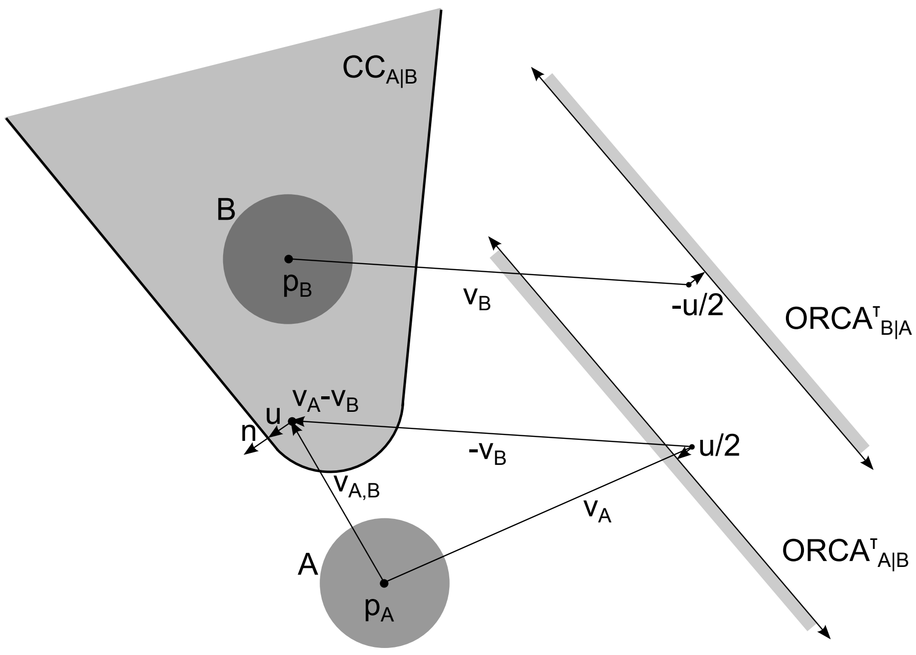


Figure 2.4: ORCA lines for agents A and B. The half-planes of permitted velocities for both agents are marked with the grey lines

assumption that every agent takes half responsibility for the collision avoidance. Formally the vector u and a set of non-conflicting velocities $ORCA^{\tau}_{A|B}$ is defined as follows (van den Berg et al., 2011):

$$u = (\underset{v \in \partial VO^{\tau}_{A|B}}{\arg\min} \|v - (v_A - v_B)\|) - (v_A - v_B), \tag{2.7}$$

$$ORCA^{\tau}_{A|B} = \{v | (v - (v_A + \frac{1}{2}u)) \cdot n \geq 0\}. \tag{2.8}$$

This way every agent calculates ORCA lines induced by every other agent and all obstacles in the environment, having a set of linear constraints on the velocity as a result. This set is formally defined as (van den Berg et al., 2011):

$$ORCA^{\tau}_{A} = D(0, v^{max}_{A}) \cap \bigcup_{B \neq A} ORCA^{\tau}_{A|B}. \tag{2.9}$$

Note that the constraint on a maximal speed of the agent v^{max}_{A} was added.

Finding a non-conflicting velocity is then a quadratic optimization task

$$\begin{aligned}
&\text{Find } v_A^{new} \text{ s. t.} \\
&\quad v_A^{new} \in ORCA_A^{\tau} \\
&\quad \left\| v_A^{new} - v_A^{des} \right\| \text{ is minimal,}
\end{aligned} \tag{2.10}$$

which given a desired velocity of the agent v_A^{des} and the $ORCA_A^{\tau}$ set of linear constraints results in a new non-conflicting velocity v_A^{new}. This optimization task can be solved by a linear program with an asymptotic time complexity $O(n)$, where n is the number of agents, because $ORCA_A^{\tau}$ is a convex region bounded by the ORCA linear constraints. The optimization function is the distance to the desired velocity v_A^{des}, which is a quadratic function, but has only one local minimum and therefore it does not invalidate the linear programming characteristics. The fact that a circular constraint for the maximal velocity is included does not alter the linear program significantly (van den Berg et al., 2011).

An implementation of the optimal reciprocal collision avoidance is available at (van den Berg et al., 2012), ORCA was successfully tested in simulations with thousands of agents.

2.1.4 Reactive Collision Avoidance for Non-Holonomic Robots

In order to make our survey more complete, in this section we will briefly mention variants of the velocity obstacle concept designed for path coordination of non-holonomic agents.

We will describe two algorithms for non-holonomic path coordination. As a model of the vehicle dynamics the first one - B-ORCA uses a bicycle model, the second one - DRCA uses a more general double integrator.

Bicycle reciprocal collision avoidance - B-ORCA (Alonso-Mora et al., 2012) builds upon the above mentioned ORCA algorithm. The main idea is that a robot with kinematic constraints is able to follow a holonomic trajectory within a certain maximum error bound. Therefore when the radius of the agents is enlarged by this error bound, we can treat the robot as holonomic and use ORCA algorithm for navigation. (Alonso-Mora et al., 2012) presents both a way of computing the error bound and a trajectory tracking controller that is able to track the holonomic path and provide continuity in agent's velocity and acceleration. B-ORCA was implemented and tested on bicycle and car models.

Distributed reactive collision avoidance - DRCA (Lalish and Morgansen, 2009) uses a more general double integrator to model the vehicle dynamics. It uses the concept of velocity obstacles to navigate vehicles in 2-d or 3-d environments. DRCA is a two step algorithm. First, the agent creates a deconfliction maneuver. This maneuver keeps the agent from colliding and results in a conflict-free state (DRCA defines a conflict as a situation where a collision will occur at some time in the future if the agents keep their velocities). Second, a deconfliction maintenance controller keeps the agents in a conflict-free state. DRCA was succesfully tested in 2-d and 3-d environments.

2.2 Planning techniques for multi-agent path coordination

Because of their online computation, reactive collision avoidance algorithms tend to find only local minima and thus are not optimal or even complete. There exist many feasible problem instances that cannot be solved by reactive techniques - see an example in Figure 4.1. These scenarios are typically solved with a global method - planning.

Planning is usually done by A* search in a discretized state space. Recently new sampling-based algorithms were proposed, which are able to find an optimal path in the continuous state space.

There are significant differences between planning and reactive collision avoidance approaches. Planning is usually centralized and offline, the coordinated paths are computed beforehand and the individual agents then follow the computed paths towards their goals. On the other hand reactive collision avoidance is decoupled and online, each agent computes a new velocity in each step of the way to its goal.

2.2.1 Sampling-based planning

The single-agent path finding problem can efficiently be solved by a well known sampling-based RRT* planning algorithm. RRT* (Karaman and Frazzoli, 2011) is based on Rapidly exploring random tree algorithm - RRT (LaValle and Kuffner, 1999), which uses fast random sampling of the state space to construct a tree in the state space rooted in the start state by connecting the newly sampled states. If a solution exists, eventually the goal state is connected to the tree and a path from the start state to the goal state can be obtained. The RRT* is an anytime extension of the RRT algorithm, it keeps searching for better solutions after the first solution is found. It has been proven that by rewiring the computed tree the RRT* algorithm is able to find solutions that almost surely converge to optimal paths and therefore it is asymptotically optimal (Karaman and Frazzoli, 2011). Even though the RRT* algorithm is in the path finding domain usually applied for single agent path finding, it can also be used for multi-agent path coordination. A redefined RRT* algorithm for multi-agent path coordination was presented in (Cap et al., 2013). We show the redefined RRT* in Algorithm 1. We will describe the RRT* for multi-agent domain in Section 2.2.2.

Algorithm 1 RRT* algorithm

```
 1: $V \leftarrow \{\mathbf{x}_{init}\}; E \leftarrow \emptyset;$
 2: while not interrupted do
 3:     $T = (V, E);$
 4:     $\mathbf{s} \leftarrow Sample();$
 5:     $\mathbf{x} \leftarrow Nearest(T, \mathbf{s});$
 6:     if $Steer(\mathbf{x}, \mathbf{s})$ then
 7:         $X_{near} \leftarrow Near(T, \mathbf{s}, |V|);$
 8:         $\mathbf{x}_p \leftarrow FindBestParent(T, X_{near}, \mathbf{s});$
 9:         $V \leftarrow V \cup \{\mathbf{x}\}; E \leftarrow E \cup \{(\mathbf{x}_p, \mathbf{x})\};$
10:         $Rewire(T, X_{near}, \mathbf{s});$
11:     end if
12: end while
```

The algorithm uses the following procedures to build a tree rooted in the start position and expand randomly in the state space C:

- *Sample* procedure returns a randomly sampled state $x \in C$.
- $Nearest(T, x)$ given a graph $T = (V, E)$ and node $x \in C$ the procedure returns a node $v \in V$ nearest to node x according to a defined metric
- $Steer(x, y)$ is a domain specific local steering procedure that given states $x, y \in C$ creates an extension from x towards y. If it is able to create such an extension, it returns $true$, otherwise it returns $false$.
- $Near(T, x, n)$ given a graph $T = (V, E)$, node $x \in C$ and number $n \in \mathbf{N}$ the procedure returns a set of nodes $\{v | v \in V \wedge dist(v, x) < \gamma(\log n/n)^{1/d}\}$, where γ is a constant and d is the dimension of the state space C.

- $FindBestParent(T, X, x)$ given a graph $T = (V, E)$, set of nodes $X \subseteq V$ and node $x \in C$ the procedure returns a node $v \in X$, which is the best parent of x, i.e. a node that yields the shortest path from the root node x_{init} to x.
- $Rewire(T, X, x)$ given a graph $T = (V, E)$, set of nodes $X \subseteq V$ and node $x \in C$ the procedure switches the parent node to x for each node $x' \in X$ for which this change would shorten the path from the root node x_{init}.

The algorithm works as follows: Until interrupted it samples the space and finds a node in the graph nearest to the new sample. Then it tries to connect these two nodes using the steering procedure. If the steering procedure fails, the algorithm samples again. Otherwise a set of nodes that are near to the new sample according to a defined metric is acquired. From this set the algorithm chooses the best new parent for the sample. Also each of the nodes in the set can possibly be rewired if the rewiring yields a shorter path to it from the root node. Even after a solution is found, the algorithm keeps sampling the state space and rewiring the tree in order to get better solutions.

2.2.2 Multi-agent sampling-based planning

The idea of applying sampling-based techniques for mutli-agent path planning was first articulated in (Cap et al., 2013), which presents a graph version of the RRT*, where two samples can be connected by the local steering procedure if it is possible to find a valid path between them by heuristic-guided greedy search in a motion graph. The multi-agent version of the algorithm then searches a joint-state space of all agents
$J = C_1 \times C_2 \times \ldots \times C_n$, where $C_i \subseteq \mathbb{R}^2$ is a state space of the i-th agent.

However, this approach is able to solve only the discretized version of the multi-agent path coordination problem where the agents' motions take place on a graph. In Chapter 4 we propose several multi-agent RRT* based algorithms that use the same idea, but are applicable for coordination of agents operating in a continous 2-d space with polygonal obstacles. We develop the idea further by incorporating the ORCA method giving rise to the ORCA-RRT* algorithm.

Chapter 3

Problem Statement

For the problem definition we first state our assumptions. We assume that the environment is fully observable i.e. the agents are equipped with sensors which provide the agents with exact positions, radii and velocities of other agents in the environment as well as the positions and shapes of all the polygonal obstacles. Further we assume a deterministic environment, where the next state of the environment is completely determined by the current state and the actions executed by the agents (Russell and Norvig, 2009). The agents' actuators are always able to move the agents to the calculated positions in the environment. Finally we assume that the agents are able to communicate with each other or a central server.

We define the multi-agent path coordination problem as follows. Consider n agents operating in the 2 dimensional Euclidean space with polygonal obstacles. The starting positions of agents are given as an n-tuple $(s_1, \ldots, s_n)$, where $s_i \in \mathbb{R}^2$ is the starting position of the i-th agent. The n-tuple $(d_1, \ldots, d_n)$ gives the agents' destinations. We assume that the agents have disc-shaped bodies, we define an n-tuple $(r_1, \ldots, r_n)$ where $r_i \in \mathbb{R}$ is the radius of the body of the i-th agent. Also we define a set $O \subset \mathbb{R}^2$ which represents the regions of the space occupied by obstacles. The final trajectory of the i-th agent $\pi_i(t)$ is a mapping $\mathbb{R} \to \mathbb{R}^2$ from the time t to the 2 dimensional Euclidean space of the agents.

The task is to find an n-tuple $(\pi_1, \ldots, \pi_n)$ such that the agents never collide and the sum of times agents spend on the path to their destinations is minimal. The problem statement is defined in Equation 3.1.

$$\begin{aligned} &\text{Find } (\pi_1, \ldots, \pi_n) \text{ s. t.} \\ &\quad CF(\pi_1, \ldots, \pi_n, O) = true \\ &\quad \sum_{i=1}^{n} t_i^d \text{ is minimal} \end{aligned} \tag{3.1}$$

The $CF(\pi_1, \ldots, \pi_n, O)$ function denotes the collision free property of the multi-agent system, it is defined in Equation 3.2. In the 2 dimensional space we use the Euclidean distance $dist_E(x, y)$. The $D(\pi_i(t), r_i)$ denotes a disc of radius r_i centered at $\pi_i(t)$. The time t_i^d is defined in Equation 3.3 as a minimal time after which the i-th agent does not leave its destination. The time t_{ALL} denotes the time when all agents reach their destinations.

$$\begin{aligned} &CF(\pi_1, \ldots, \pi_n, O) = true \text{ iff} \\ &\quad \forall i, j, t, i \neq j : dist_E(\pi_i(t), \pi_j(t)) \geq r_i + r_j \\ &\quad and \\ &\quad \forall i, t, o \in O : D(\pi_i(t), r_i) \cap o = \emptyset \end{aligned} \tag{3.2}$$

$$t_i^d = \min(t_{i,x} | \forall t \in \langle t_{i,x}, t_{ALL} \rangle : \pi_i(t) = d_i) \tag{3.3}$$

We define a quality of a solution of the multi-agent path coordination problem in the means of suboptimality, using an idealistic solution cost, as follows:

- **Idealistic solution cost**, cost of a solution for which the CF function in equation 3.2 is relaxed in a way that it omits its first condition i.e. permits collisions between agents. It's a sum of goal arrival times of single agents,

$$t_{idealistic} = \sum_{i=1}^{n} t_i^d. \tag{3.4}$$

  The goal arrival time t_i^d of the i-th agent is computed using the shortest path from s_i to d_i on a visibility graph and assuming the speed of agents to be their maximal speed.

- **Suboptimality** shows how many times the solution is worse than the idealistic solution,

$$suboptimality = \frac{\sum_{i=1}^{n} t_i^d}{t_{idealistic}}. \tag{3.5}$$

Chapter 4

Proposed algorithm

In this chapter we first analyze the reactive collision avoidance algorithm we chose as the most promising state of the art method for the multi-agent path coordination - ORCA (van den Berg et al., 2011). We will focus on its weaknesses in Section 4.1. Then in Section 4.2 we formalize the use of the RRT* for multi-agent planning and propose two RRT* based algorithms - Line-RRT* and VisibilityGraph-RRT*. The purpose of these algorithms is first to study the applicability of the RRT* for multi-agent planning in continous environments and second to provide a comparison methods based on the RRT* for the proposed algorithm - ORCA-RRT*. We present our ORCA-RRT* algorithm in Section 4.3. Finally in Section 4.4 we describe how the ORCA-RRT* could be deployed on a team of robots.

4.1 Weaknesses of the Optimal Reciprocal Collision Avoidance

As we mentioned in Chapter 2, most of the reactive collision avoidance methods including ORCA are applied online i.e. during the actual motion of the agents. This approach can be observed as greedy behavior since the agents decide about their new velocities in each step without considering the future states of the environment. Due to the greediness of the online reactive collision avoidance methods there is no guarantee that the agents will reach their goals. A well known disadvantage of the greedy approach is its inability to escape local minima. In the multi-agent path coordination problem a local minimum is a situation where no agent is able to further decrease its distance to goal, but the overall goal state is not reached. This happens for example when the agents with opposite goals are unable to exchange their positions. Even if the agents eventually escape such a minimum and reach their goals, the resulting solution is suboptimal. Also note that ORCA is only locally optimal - in each step the agents choose optimal velocities with respect to the velocity constraints and the desired velocity. This however does not imply that the algorithm would return optimal trajectories.

There are many instances of the multi-agent path coordination problem that ORCA is not able to solve due to the lack of higher-level cooperation. An example of such an instance is any scenario where an agent has to let another agent move through a certain part of the environment first, before it can continue moving towards its goal. In Figure 4.1 we show an example of a problem instance that cannot be solved by ORCA reactive technique, but is solved by the ORCA-RRT* algorithm. Observe how ORCA gets stuck in the local minimum.

4.2 Proposed Multi-agent RRT* approach

In this section we first define the multi-agent RRT* approach for multi-agent path coordination. Then we provide specific multi-agent RRT* based algorithms.

The general RRT* algorithm was revealed in Algorithm 1. We define the following attributes of the algorithm that allow its use in the multi-agent domain:

- **State space** of the RRT* planning is defined for n agents as:

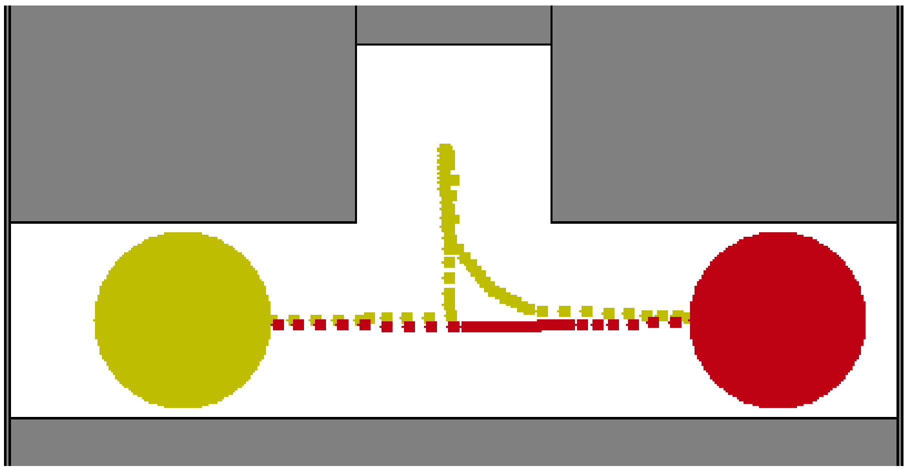
  $$J = C_1 \times C_2 \times \ldots \times C_n, \tag{4.1}$$

  where $C_i \subseteq \mathbb{R}^2$ is a state space of the i-th agent. We call J a joint-state space. Since $C \subseteq \mathbb{R}^2$, for the joint-state space it holds that $J \subseteq \mathbb{R}^{2n}$. A state $\mathbf{x} \in J$ is then given as

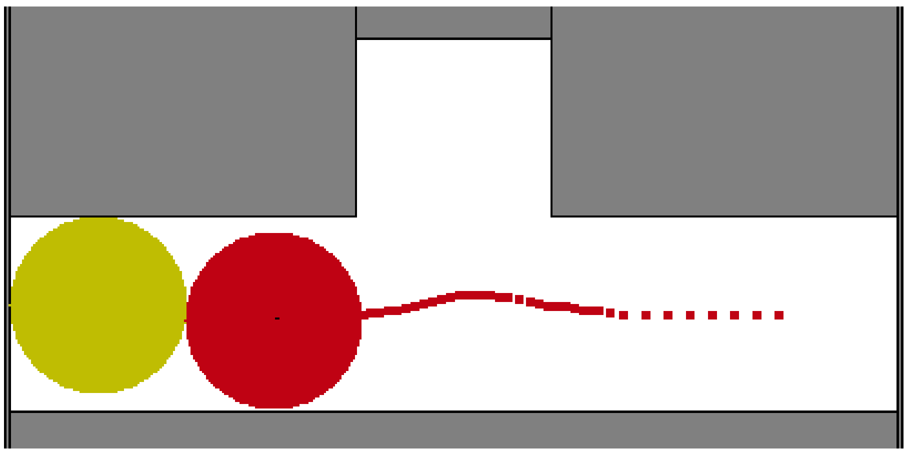
  $$\mathbf{x} = ((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)), \tag{4.2}$$

  where $(x_i, y_i) \in C_i$ is a position of the i-th agent. We call such a state a joint state.

  Note that other possible representation of the state space could include a time variable for each state $x \in C$. This would increase the dimension of the search space to $J_t \subseteq \mathbb{R}^{3n}$ and therefore make the multi-agent path coordination problem harder to solve. On the other hand our definition of the joint-state space brings



(a) Solution found by ORCA-RRT*



(b) Partial solution found by ORCA. ORCA is unable to escape a local minimum

Figure 4.1: Example of an instance solved by ORCA-RRT* that cannot be solved by ORCA (two disk shaped agents exchange their positions, points show trajectories $\pi_i(t)$)

a significant aspect to the resulting solution of the problem - since the times the agents spend moving to their end positions in each extension differs and we require all agents to be in their end positions at the end of the extension, the faster agents have to wait in their end positions until the slower agents arrive at theirs. This increases the cost of the solution.

- **Distance function** $dist(\mathbf{x_1}, \mathbf{x_2})$ gives a lower bound estimate on the distance between the two joint states $\mathbf{x_1}, \mathbf{x_2}$. We use the following distance function:

$$dist(\mathbf{x_1}, \mathbf{x_2}) = \sum_{i=1}^{n} \frac{dist_E(\mathbf{x_1}(i), \mathbf{x_2}(i))}{v_i^{max}} \tag{4.3}$$

where $dist_E(\mathbf{x_1}(i), \mathbf{x_2}(i))$ is the Euclidean distance between positions of the i-th agent in states $\mathbf{x_1}$ and $\mathbf{x_2}$ and v_i^{max} is the maximal speed of the i-th agent. Note that this way the distance expresses a sum of minimal times in which the agents can move between the states assuming no conflicts occur.

- **Steering procedure** $Steer(\mathbf{x_1}, \mathbf{x_2})$ is an essential part of the RRT* algorithm that defines a way in which the states are connected to form the search tree - see Algorithm 1, Line 6. We call the output of the steering procedure an extension from $\mathbf{x_1}$ to $\mathbf{x_2}$.

  We define the steering function as

$$Steer(\mathbf{x}, \mathbf{y}) = \begin{cases} E(\mathbf{x}, \mathbf{y}, O) & \text{if} \\ & \quad \{\pi_i\} = E(\mathbf{x}, \mathbf{y}, O) \\ & \mathit{and} \quad CF(\{\pi_i\}, O) \\ & \mathit{and} \quad \forall i : \pi_i(t_{max}) = y_i \\ \emptyset \quad \text{otherwise} & \end{cases} \tag{4.4}$$

  where $\mathbf{x}, \mathbf{y} \in J$ and $\pi_i(t_{max})$ returns the last point in the trajectory of the i-th agent. The function $E(\mathbf{x}, \mathbf{y}, O)$ denotes the extension and is defined as $E(\mathbf{x}, \mathbf{y}, O) \to \pi_1 \times \ldots \times \pi_n$, we assume that $\forall i : \pi_i(0) = x_i$. The $CF(\{\pi_i\}, O)$ function which defines the collision free property of the multi-agent system, was defined in Chapter 3. Note that this definition specifies the conditions that an extension has to satisfy in order to be accepted as valid. We propose three methods for the extensions. In an ascending order of complexity those are Line-RRT*, VisibilityGraph-RRT* and ORCA-RRT*.

4.2.1 Line-RRT*

The first extension method is Line-RRT*. The agents are allowed to move between states only using straight line trajectories. The speed of the agents is considered constant. The Line-RRT* extension is formally defined as follows:

$$E_{Line}(\mathbf{x}, \mathbf{y}, O) = (\pi_1, \dots, \pi_n) \qquad (4.5)$$
$$\text{where } \pi_i = line(x_i, y_i).$$

The $line(x, y) : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \Pi$ is a function defined as

$$\begin{aligned} & line(x_i, y_i) \rightarrow \{\pi_i\} : \\ & \pi_i(t) = \begin{cases} x_i + v_i \cdot t \frac{y_i - x_i}{|y_i - x_i|} & \text{for } t < \frac{|y_i - x_i|}{v_i} \\ y_i & otherwise \end{cases} \end{aligned} \qquad (4.6)$$

where v_i denotes the speed of the i-th agent.

4.2.2 VisibilityGraph-RRT*

The Line-RRT* extension does not use any information about the static obstacles in the environment. Therefore many extensions computed by the Line-RRT* are rejected (see Equation 4.4) because of the second condition in the CF function in Equation 3.2 which requires that the trajectories do not intersect with any obstacle. In the VisibilityGraph-RRT* extension we therefore include a specific navigation that connects the start and end positions of the agents in each extension in a way that satisfies the CF function. For this task we use a visibility graph navigation.

The visilibity graph is a graph of so called intervisible locations. This means that whenever an edge connects two vertices, this edge does not intersect any obstacle. The vertices in the graph correspond to the vertices of the obstacles in the environment, which are moved away from the obstacles by the radii of agents' bodies. The visibility graph is constructed using vertices created by inflating each obstacle. We also add start and goal positions of all agents. We then add an edge between any two vertices if it does not intersect with any obstacle. Finally the shortest path for each agent is obtained using an A* search in the visibility graph. This way we use the information about positions and shapes of the obstacles to deal with the obstacle constraint in Equation 3.2. Single agent planning in an environment with polygonal obstacles using visibility graph algorithm is complete and optimal (LaValle, 2004).

VisibilityGraph-RRT* extension is obtained by applying visibility graph navigation for obstacle avoiding. The agents move on a piece-wise linear path composed of one or more line segments. VisibilityGraph-RRT* extension is formally defined as

$$E_{VisibilityGraph}(\mathbf{x}, \mathbf{y}, O) = (\pi_1, \dots, \pi_n) \quad (4.7)$$
$$\text{where } \pi_i = lineSegments(x_i, y_i).$$

Let the shortest path of the i-th agent in the visibility graph from a start vertex to an end vertex (which denotes the start and end positions of the i-th agent) be denoted as a sequence of edges $(e_{i1}, \dots, e_{im_i})$. The $lineSegments(x, y) : \mathbb{R}^2 \times \mathbb{R}^2 \to \Pi$ is a function defined as

$$lineSegments(x_i, y_i) \to \{\pi_i\} : \quad (4.8)$$
$$\pi_i(t) = \begin{cases} source(e_{ij}) + v_i \cdot t \frac{e_{ij}}{|e_{ij}|} \\ \quad \text{for } t \in \left(\frac{\sum_{k=1}^{j-1} |e_{ik}|}{v_i}, \frac{\sum_{k=1}^{j} |e_{ik}|}{v_i}\right), j \in \langle 1, m_i \rangle \\ y_i \quad otherwise \end{cases}$$

where the $source(e)$ function returns the source vertex of the edge e.

4.2.3 Weaknesses of the Multi-agent RRT* approach

As mentioned above, the multi-agent RRT* planning takes place in a high dimensional joint-state space $J \in \mathbb{R}^{2n}$. Therefore with the increasing number of agents n the dimensionality of the joint-state space grows exponentially. This fact implies a large computational time, which can make the algorithm unfeasible for any real world application. Our experiments in Chapter 6 show that the Line-RRT* and VisibilityGraph-RRT* algorithms, when given the computational time limit of 5 seconds, are able to solve the multi-agent path coordination problem for up to around 5 agents.

The reason for this drawback is the fact that the majority of the computed extensions is rejected by the CF function (see Equations 4.4 and 3.2), because the Line-RRT* does not consider any collision avoidance and the VisibilityGraph-RRT* only considers collision avoidance with static obstacles. It would therefore be convenient to propose an algorithm that would consider collision avoidance with both static obstacles and other agents in the environment. Such an algorithm could significantly increase the proportion of computed extensions that are collision free according to Equations 4.4 and 3.2 in the set of all computed extensions.

We propose an algorithm that is able to produce extensions which keep the collision free property, the ORCA-RRT* algorithm.

4.3 Proposed ORCA-RRT* algorithm

We propose a new algorithm for solving the multi-agent path coordination problem. Its challenge is to overcome the above mentioned disadvantages of both sampling-based planning and reactive collision avoidance algorithms in order to increase the coverage of problem instances. ORCA-RRT* combines the planning algorithm - RRT* with the reactive collision avoidance algorithm - ORCA, in the following fashion. A central solver runs the RRT* algorithm offline. As a way of connecting the randomly sampled nodes the steering procedure uses ORCA for simulation of motions of the multiple agents. Note that this way the reactive collision avoidance runs as a simulation instead of running online, as it is mostly used. After a given time limit, if a solution is found, it can be used for navigation of the physical agents towards their goals. In Section 4.4 we propose a deployment of the ORCA-RRT* algorithm on a team of robots.

We described the ORCA algorithm in Chapter 2. While ORCA guarantees collision free motions of the agents, it does not guarantee that the agents will ever reach their goals. A controller has to be designed that is responsible for navigating an agent to its goal position by providing a desired velocity vector for the ORCA algorithm in each of its steps. If there were no static obstacles in the environment, such a controller could simply return a vector pointing in the direction of the goal. Since we assume static obstacles in the environment, we propose a more complex controller - a visibility graph based controller.

The visibility graph based controller works as follows. The first step is a construction of the visibility graph. This is done identically as for the VisibilityGraph-RRT* (see Section 4.2.2). After that each agent assigns costs to all vertices in the visibility graph based on their distance from its goal using the Dijkstra algorithm. In each step of the ORCA algorithm the visibility graph based controller finds a best vertex in the graph i.e. the vertex which cost added up to the distance from agents current position is minimal. Also the line between agent's position and the best vertex cannot intersect with any obstacle. Finally the visibility graph based controller returns the desired velocity vector which points to this best vertex.

The RRT* algorithm mentioned in Algorithm 1 used in ORCA-RRT* has the following specific features:

- The *Sample* procedure in the first iteration of the RRT* algorithm returns the goal state. In the following iterations it randomly samples the space with some higher than average probability of sampling the goal state.

- The *Steer* procedure (the local steering procedure) will run ORCA with the following definition of the stopping criterion: The ORCA steering procedure stops if either it reaches the goal state or the suboptimality of the computed partial solution is higher than a suboptimality threshold α.
- Suboptimality threshold α is an upper bound on the suboptimality of a solution. If the suboptimality is higher, the solution is rejected. The behavior of ORCA-RRT* changes significantly for different suboptimality thresholds because it determines the maximal length of the ORCA-RRT* extension.

This way we make sure that the ORCA-RRT* algorithm first runs the ORCA algorithm as long as the solution is good enough with respect to the suboptimality threshold α. This fact provides a good theoretical properties of the ORCA-RRT* algorithm which we discuss in Chapter 5.

The ORCA-RRT* extension function is formally defined as

$$E_{ORCA}(\mathbf{x}, \mathbf{y}, O) = (\pi_1, \ldots, \pi_n), \tag{4.9}$$

where π_i is a trajectory of the i-th agent obtained by ORCA method with $\mathbf{x}$ as start positions and $\mathbf{y}$ as goal positions.

The ORCA-RRT* extension is exposed in Algorithm 2, the steering procedure takes on the input start and goal states $\mathbf{x_s}, \mathbf{x_d} \in J$, where J is the joint-state space (see Section 4.2), and returns a solution - an n-tuple $(\pi_1 \ldots \pi_n)$ of trajectories. Algorithm 1 - the RRT* algorithm along with Algorithm 2 - the ORCA extension, show the proposed ORCA-RRT* algorithm.

Algorithm 2 ORCA-RRT* extension

```
1: Steer($\mathbf{x_s}$, $\mathbf{x_d}$){
2: setupScenario($\mathbf{x_s}$, $\mathbf{x_d}$)
3: solution $\leftarrow \emptyset$
4: suboptimality $\leftarrow 0$
5: while goal not reached by all agents and suboptimality $< \alpha$ do
6:     for each agent do
7:         desired velocity $\leftarrow$ visibility graph based controller
8:         compute ORCA lines
9:         new velocity $\leftarrow$ linear program with desired velocity as the objective function
           and ORCA lines as linear constraints
10:        agent's position $\leftarrow$ agent's position + new velocity $\cdot$ time step length
11:    end for
12:    solution $\leftarrow$ concatenate agents' positions to the solution
13:    suboptimality $\leftarrow$ compute suboptimality of the partial solution
14: end while
15: return solution }
```

4.4 ORCA-RRT* deployment

In this section we describe the way the proposed ORCA-RRT* algorithm could be deployed on a team of real robots. A use case can for instance be the Automatic Guided Vehicle Systems. We have following assumptions about the robots:

- Each robot is equipped with sensors which are able to provide its position in the given environment
- Each robot is able to communicate (send and receive messages) with all other robots in the team
- Robots possess information about positions and shapes of all the obstacles in the environment
- At least one of the robots is able to run the computation of the ORCA-RRT* algorithm

- Robots are able to navigate on a trajectory $\pi(t)$ (see Chapter 3)
- Robots have disk shaped bodies (or their bodies can be circumscribed by a disk) and are aware of the radius of the disk

Given these assumptions the ORCA-RRT* algorithm can be deployed on a team of robots as follows. Consider an environment occupied by a number of holonomic robots and static obstacles. In a situation where any of the robots is not at its goal position and does not have a trajectory to follow, it asks all the other robots to send it their current and goal positions. The robot also needs information about the radii and maximal velocities of all robots. This can be either stored in its memory since it is constant or obtained by communication. This robot then becomes a central server and runs the ORCA-RRT* algorithm with the obtained parameters. Output of the ORCA-RRT* algorithm is a set of non-conflicting trajectories $\pi(t)$ for each robot in the team. Once computed, the robot sends the trajectories to all other robots. Once a robot receives a trajectory, it starts following it. Note that the trajectory specifies the position the robot has to be at in the given time. A collision free motion is guaranteed for as long as the robots keep following the trajectories and do not change their goals. Once a robot reaches its goal, it by default stays at the goal location. If it acquires a new goal, it will become a central server and the process is repeated.

The ORCA-RRT* is a centralized algorithm, which can be computed by any of the robots or some other central server. Recently (Otte and Correll, 2010) proposed an RRT based algorithm that distributes the computation load among all robots in the team. Each robot builds its own tree and shares found solutions with other robots. The other robots then prune the branches with higher costs than the minimal cost of shared solutions. This approach is called Any-Com Intermediate Solution Sharing. It can be used in the deployment of the ORCA-RRT* algorithm in order to reach lower computational times.

The main difference of our way of deploying the ORCA-RRT* algorithm and any reactive collision avoidance technique is that the ORCA-RRT* is finding the coordinated paths offline, before the actual robots motion takes place. It then provides trajectories that the robots follow. In reactive collision avoidance techniques such as ORCA the trajectories are not known beforehand, the robots always decide about their motion one step ahead only.

Chapter 5

Theoretical Analysis

In this Chapter we theoretically analyze the proposed ORCA-RRT* algorithm. First we provide a time complexity analysis of the algorithm. Then we prove that ORCA-RRT* has better coverage of the instance space than the ORCA reactive technique. We also prove this hypothesis experimentally in Chapter 6.

5.1 Time Complexity Analysis

In this Section we provide an average-case time complexity analysis of the proposed algorithms. We first provide the time complexity of the multi-agent RRT* algorithm and the proposed steering procedures. Then we show the time complexity of the two multi-agent RRT* based algorithms and the ORCA-RRT*.

The time complexity is a function of a number of agents n, a number of obstacles m, a time length of trajectories t and a number of randomly sampled states s. We show the time complexity analysis of the multi-agent RRT* in Table 5.1 and of proposed steering procedures in Table 5.2. We note that in the RRT* algorithm a depth first search algorithm is used, which given that we use a KD tree has the average-case time complexity $O(\log s)$ for s nodes in the tree (Bentley, 1975), see Table 5.1. Each item in the Algorithm column in Table 5.2 corresponds to a term in the Time complexity column. Both columns are ordered. For the complexity of the linear program of the ORCA algorithm we refer to (van den Berg et al., 2012).

Since RRT* is an anytime algorithm, we show the time complexity of one iteration instead of the time complexity of the whole algorithm. The time complexity of one iteration of the proposed algorithms is in Table 5.3.

In the time complexities in Table 5.3 several terms occur. A collision checking procedure is composed of mutual collision checking of agents' trajectories, which has asymptotic complexity $O(s \cdot n^2 \cdot t)$ - we check at most s times the pairs of trajectories of length t of n agents, and collision checking of trajectories with obstacles with the complexity $O(s \cdot n \cdot m \cdot t)$ - we check at most s times n trajectories of length t with m obstacles. The ORCA-RRT* does not require collision checking, but the asymptotic complexity of the linear program is also $O(s \cdot n^2 \cdot t)$ (van den Berg et al., 2012). Note that the VisibilityGraph-RRT* does not have to check collisions with obstacles because the visibility graph algorithm already provides a path not conflicting with any obstacle. We can therefore omit the second condition in the CF function in Equation 3.2. Finding the shortest path in the visibility graph using a Dijkstra algorithm has complexity $O(s \cdot n \cdot m \log m)$ (Fredman and Tarjan, 1984) and the complexity $O(s \cdot n \cdot m^2 \cdot t)$ corresponds to choosing the desired velocity in the ORCA algorithm, because we have to do collision checking with m obstacles for the nodes in the visibility graph (see Section 4.3).

Procedure	Algorithm	Time Complexity
$Sample$	Random sampling	$O(1)$
$Nearest$	Depth first search with KD Tree	$O(log(s))$
$SteeringProcedure$	Line/VisibilityGraph/ORCA-RRT*	see table 5.2
$Near$	Depth first search with KD Tree	$O(log(s))$
$FindBest$	Steering procedure	$O(s \cdot e)$
$Rewire$	Depth first search with KD Tree, Steering procedure	$O(log(s) \cdot e)$

Table 5.1: Time complexity of Multi-agent RRT* procedures (with respect to the number of randomly sampled states s)

Extension	Algorithm	Time Complexity $O(e)$
Line-RRT*	Collision checking with agents and obstacles	$O(n^2 \cdot t + n \cdot m \cdot t)$
VisibilityGraph-RRT*	Find shortest paths Collision checking with agents	$O(n \cdot m \log m + n^2 \cdot t)$
ORCA-RRT*	ORCA • Evaluate visibility graph • Linear program, compute agents ORCA lines • Find desired velocity, compute obstacle ORCA lines	$O(n \cdot m \log m +$ $+ n^2 \cdot t + n \cdot m^2 \cdot t)$

Table 5.2: Time complexity of Multi-agent RRT* steering procedures (with respect to the number of agents n, the number of obstacles m and the time length of the trajectories t)

Algorithm	Time Complexity
Line-RRT*	$O(s \cdot n^2 \cdot t + s \cdot n \cdot m \cdot t)$
VisibilityGraph-RRT*	$O(s \cdot n^2 \cdot t + s \cdot n \cdot m \log m)$
ORCA-RRT*	$O(s \cdot n^2 \cdot t + s \cdot n \cdot m \log m + s \cdot n \cdot m^2 \cdot t)$

Table 5.3: Time complexity of one iteration of the proposed algorithms (with respect to the number of randomly sampled states s, the number of agents n, the number of obstacles m and the time length of the trajectories t)

5.2 Problem Instances Coverage

We first provide the following definitions of a specific type of solutions of the multi-agent path coordination problem, the coverage of the problem instance space itself and a specific type of the coverage - the α-coverage.

Definition 5.1: $\boldsymbol{\alpha}$-solution of a problem instance is a solution with a suboptimality at most α.

Definition 5.2: Coverage of a problem instance space given by an algorithm is a set of instances, which are solved by the algorithm.

Definition 5.3: $\boldsymbol{\alpha}$-coverage of a problem instance space given by an algorithm is a set of instances, which are solved by the algorithm and the solutions are α-solutions.

Recall how we defined the suboptimality in Chapter 3 and the suboptimality threshold α in Section 4.3:

- Suboptimality shows how many times the solution is worse than the idealistic solution,
$$suboptimality = \frac{\sum_{i=1}^{n} t_i^d}{t_{idealistic}}\,. \tag{5.1}$$
- Suboptimality threshold α is an upper bound on the suboptimality of a solution. If the suboptimality is higher, the solution is rejected.

The α-coverage is an important concept, since we are often interested only in solutions, which are good in terms of quality or suboptimality. Note that with no suboptimality threshold the coverage set and the α-coverage set are equal. We assume that for the multi-agent path coordination problem there always exists a solution.

Theorem 5.1 (α-coverage of ORCA-RRT* and ORCA): *Given a multi-agent path coordination problem, the α-coverage of the problem instance space given by ORCA-RRT* is a superset of the α-coverage given by ORCA. Moreover when given a sufficient running time limit, the α-coverage given by ORCA-RRT* is a proper superset of the α-coverage given by ORCA.*

Proof (Theorem 5.1): To prove this theorem, we first have to prove that for any instance, for which the ORCA algorithm finds an α-solution, the ORCA-RRT* finds an α-solution too. Further we prove that there exist instances for which ORCA does not find an α-solution, while ORCA-RRT* does.

The first iteration of the ORCA-RRT* algorithm is equivalent to running the ORCA algorithm as long as the suboptimality of the partial solution is lower than the suboptimality threshold α (see Section 4.3). This way if an α-solution exists which the ORCA algorithm would find, the ORCA-RRT* algorithm will find it too. Due to the anytime property of the RRT* algorithm the ORCA-RRT* keeps searching the state space even when the first extension is not successful i.e. if the ORCA algorithm would fail to find an α-solution. Note that this can happen since ORCA is not an optimal or complete algorithm. Since there always exists a solution, it can be with a probability higher than zero sampled by the *Sample* procedure of the RRT* algorithm. Therefore if the ORCA algorithm fails, the ORCA-RRT* algorithm will still with a probability higher than zero find a solution. $\square$

Chapter 6

Experimental Analysis

We test four algorithms for the multi-agent path coordination problem. Three RRT* based - Line-RRT*, VisibilityGraph-RRT* and ORCA-RRT*, and one reactive collision avoidance algorithm - ORCA. In this chapter we first describe the test setting - test inputs that define the problem instance space and a benchmark set, a subset of the problem instance space which we used for testing. We then define the measured test outputs and finally we present the results along with their evaluation.

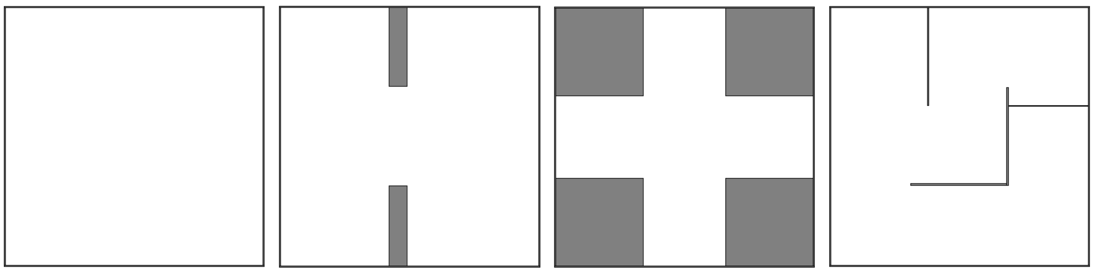
6.1 Test inputs

The test inputs define the multi-agent path coordination problem instance space. The following parameters can be set to obtain different problem instances:

- **Environment** - our problem instance space contains four different environments depicted in Figure 6.1. We designed these environments as representative examples of difficult regions in common real world environments. The size of the environments is 1000×1000.
- **Number of agents** is ranging from one to ten in our problem instance space. We chose this selection because the RRT* based algorithms (Line-RRT* and VisibilityGraph-RRT*) have very low success rate in higher dimensions. Note that Theorem 5.1 states that the ORCA-RRT* algorithm is able to solve any instance that ORCA is able to solve. Since ORCA performs well for up to several hundreds of agents (van den Berg et al., 2011), ORCA-RRT* would be able to solve instances

with higher number of agents as well.

- **Agent body radius** is ranging from 50 to 100. This is a significant parameter since it determines whether the agents are able to exchange positions in narrow corridors. Figure 6.2 shows the minimal and maximal radii of the agents along with optimal single agent trajectories obtained by the visibility graph algorithm.
- **Suboptimality threshold** α determines how many times worse the cost of the solution can be than the idealistic solution cost. If the cost is higher, the solution is rejected. Recall the definition of the suboptimality threshold from Section 4.3. Note that while the behavior of Line-RRT*, VisibilityGraph-RRT* and ORCA does not depend on the suboptimality threshold, the behavior of ORCA-RRT* changes significantly for different suboptimality thresholds because it determines the maximal length of the ORCA-RRT* extension. We chose the following values for the suboptimality threshold: 2.5, 5, 10 and 1000, where 1000 was used for problem instances with no suboptimality threshold.
- **Running time limit** is set to 5 seconds, but we also provide results from tests with 1 second running time limit in order to show how the performance of the anytime RRT* based algorithms depends on this limitation. We note that the experiments were performed on AMD FX(tm)-8150 with 16 GB RAM.



(a) Empty environment (b) Door environment (c) Cross environment (d) Maze environment

Figure 6.1: Problem environments

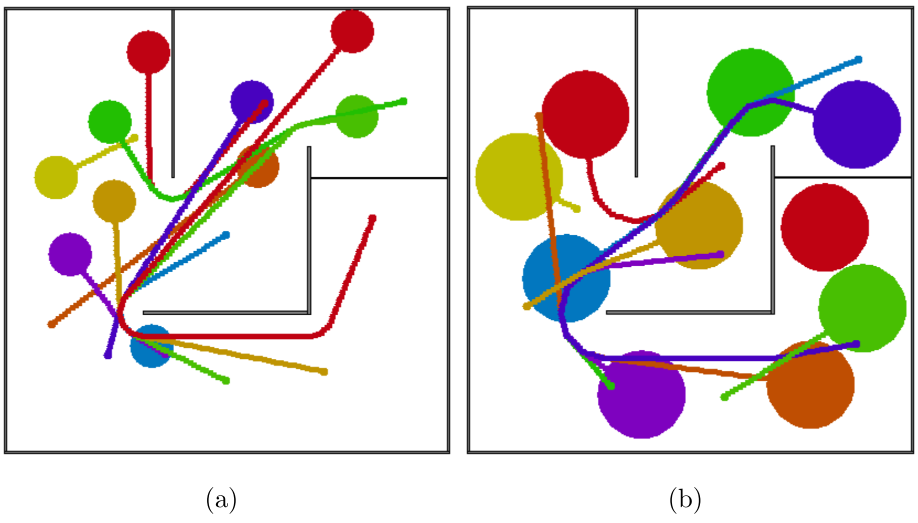


Figure 6.2: Example of the maze environment - 10 agents with radii a) 50, b) 100, single agent optimal paths

6.2 Benchmark set

We produced a benchmark set of various scenarios for testing. For each combination of the environment, the number of agents, the agent body radius and the suboptimality threshold the benchmark set contains 10 different settings of agents' start and goal positions.

We ran the ORCA algorithm 2400 times. Since RRT* is a stochastic algorithm, we ran the RRT* based algorithms 10 times on each problem instance with different seeds. We ran Line-RRT* and VisibilityGraph-RRT* each 24000 times. Because the behavior of the ORCA-RRT* algorithm depends on the chosen suboptimality threshold, we ran the ORCA-RRT* 4 times for each instance with different suboptimality thresholds. Altogether this would mean 146400 executions, but some problem instances were impossible to create due to the combination of high number of agents and large agent body radius. Therefore altogether we executed 144212 runs.

The benchmark set was created by a Benchmark set generator which guarantees that for each problem instance there is exactly one conflict cluster i.e. the path finding problem cannot be divided into mutually non-conflicting groups of agents, which would be easier to solve. This is guaranteed by adding agent's random start and goal positions iteratively only when a conflict occurs between agent's shortest path from start to goal and any other agent's shortest path. The procedure that creates a problem instance is given in Algorithm 3.

Algorithm 3 Benchmark set generator: Create problem instance

```
1: for i in 1:numberOfAgents do
2:     colliding = false
3:     while not colliding do
4:         start = randomSample
5:         goal = randomSample
6:         path = findShortestPath(start, goal, obstacles)
7:         colliding = findConflict(path, allPaths)
8:     end while
9:     allPaths.add(path)
10:    starts.add(start)
11:    goals.add(goal)
12: end for
```

6.3 Test outputs

We measure several parameters that provide different views on the behavior of the algorithms and their ability to solve the given problem instances. The measured parameters are:

- **Idealistic solution cost**, cost of a solution for which the CF function in equation 3.2 is relaxed in a way that it omits its first constraint i.e. permits collisions between agents. We defined the idealistic solution cost in Equation 3.4.
- **Suboptimality** shows how many times the solution is worse than the idealistic solution. We defined the suboptimality in Equation 3.5.
- **Success rate** shows the percentage of scenarios solved by the algorithms. The success rate depends on the suboptimality in a way that a solution is successful only if its suboptimality is lower than α, the suboptimality threshold defined in Section 4.3.
- **α-coverage** was defined in Section 5.2. It relates to the success rate in a way that it is a set of successful solutions (successful in terms of the suboptimality and the suboptimality threshold α). Note that with no suboptimality threshold the coverage set and the α-coverage set are equal.

6.4 Evaluation

In this section we provide an evaluation of our experiment results. First we show how the proposed ORCA-RRT* algorithm behaves on one specific problem instance. Then we compare the algorithms based on their success rate. Finally we assign ranks to the algorithms based on the suboptimalities of their solutions.

6.4.1 ORCA-RRT* solution

We picked a problem instance - the cross environment occupied by 6 agents of body radius 50 with a suboptimality threshold 10 and running time limit 5 seconds. On this instance we show the behavior of the ORCA-RRT* algorithm. In Figure 6.3 we show the solutions that the ORCA-RRT* provides during the 5 second running time. In Table 6.1 we show the times, iteration numbers and suboptimalities of the solutions. Note that the suboptimality is monotonically decreasing with the increasing solution number. This is a property of the RRT* algorithm. Also note that the first solution was found in the 18th iteration. This means that given the suboptimality threshold the ORCA algorithm is not able to provide a solution. Otherwise the ORCA-RRT* algorithm would according to Theorem 5.1 find the first solution in the first iteration. In Figure 6.4 we show time snapshots of the simulation where the agents follow the trajectories from the best solution from Figure 6.3.

6.4.2 Success rate of the algorithms

In this section we compare the algorithms by their success rate for various numbers of agents, agent body radii, environments and suboptimality thresholds. Note that the sucess rate closely relates to the concept of α-coverage. The most important experiment results in this thesis are depicted in Figures 6.5, 6.6, 6.7 and 6.8. In these figures we show the successrate for different suboptimality thresholds given the number of agents and the agents' bodies radii. Each of these figures shows results for different suboptimality threshold. Three significant phenomena occur.

First, the success rate of RRT* based algorithms Line-RRT* and VisibilityGraph-RRT* drops fast with increasing number of agents. This behavior was expected because the planning takes place in a state space exponential to the number of agents. These algorithms are able to solve the defined problems for the number of agents up to approximately 5.

Second, the success rate of the ORCA reactive technique drops with increasing agents' radii. This is due to the nature of the reactive algorithm. The problem instances contain local minima - states, where agents get stuck close to each other because they only optimize their single agent behavior. Since reactive technique can be observed as greedy approach, it is often unable to overcome these local minima. The significance of this behavior increases in more cluttered environments, see for instance figures 6.8(c), 6.8(d), where by increasing the radii of agents' bodies we eventually make the corridors narrow in a way that the agents are unable to swap their positions without leaving the corridor. Such a situation is hard to solve locally.

Furthermore, the solutions provided by ORCA often have high suboptimality. A lower suboptimality threshold significantly decreases the success rate - notice the difference between Figures 6.5(a) and 6.6(a). This typically happens in cluttered environments, where the agents are likely to get stuck in slowly evolving deadlock situations.

Third, the success rate of the ORCA-RRT* algorithm is close to one for both high number of agents and high agent radius. It drops only with the combination of high extremes of both parameters. This behavior is achieved by the combination of planning and reactive approaches. The planning part is able to solve problem instances containing local minima, while the reactive part is able to solve the poblem instances with high number of agents. We can observe for example in Figure 6.8(c) that while the other algorithms are only successful in parts of the instance space, the ORCA-RRT* benefits from both its parts by covering the union of their α-coverage.

Furthermore, the solutions of the ORCA-RRT* algorithm often have lower suboptimality than the solutions of ORCA, which we can observe in slower deterioration of the α-coverage with decreasing suboptimality threshold. Since the first iteration of the ORCA-RRT* is identical to running ORCA with the suboptimality threshold α as an upper bound for the solution suboptimality, the slower deterioration shows the ability of the ORCA-RRT* algorithm to improve its solutions over time.

Finally, all our experimental data agrees with the Theorem 5.1, which states that the α-coverage of the ORCA-RRT* is never worse than the α-coverage of ORCA, but it can be better. Observe this phenomenon in Figures 6.5, 6.6, 6.7 and 6.8. Note that the Figure 6.8 relates to the coverage of the instance space (Definition 5.2) without the bound on the suboptimality in the form of the suboptimality threshold α.

Figure 6.9 and Tables 6.2 and 6.3 show the success rate separately for number of agents and agents' body radii, given the suboptimality threshold $\alpha = 2.5$. Figure 6.10 and Tables 6.4 and 6.5 show the same data for no suboptimality threshold. This view on the data shows three significant phenomena.

First, the success rate of the VisibilityGraph-RRT* algorithm is often lower than the success rate of the Line-RRT*. This fact is not intuitive, since we expected that with the better use of environment knowledge (VisibilityGraph-RRT* uses information about obstacles to create trajectories with no obstacle intersections) the success rate would increase. The reason we believe is behind the lower success rate of the VisibilityGraph-RRT* is a high overhead that comes with searching for trajectories with no obstacle intersections as opposed to fast random sampling of the state space and simple line trajectories computed by the Line-RRT*.

Second, the success rate of the ORCA algorithm depends significantly on the suboptimality threshold. With no suboptimality threshold, the success rate of ORCA is almost always better than the success rate of Line-RRT* and VisibilityGraph-RRT* (see Figure 6.10). On the other hand, with the low suboptimality threshold the success rate of ORCA is lower than the success rate of all other measured algorithms (see Figure 6.9). Reason for this fact is the above mentioned high suboptimality of the solutions provided by the ORCA algorithm.

Third, the success rate of the ORCA-RRT* algorithm is higher for all numbers of agents and agents' bodies radii. This is again thanks to the combination of planning and reactive approach.

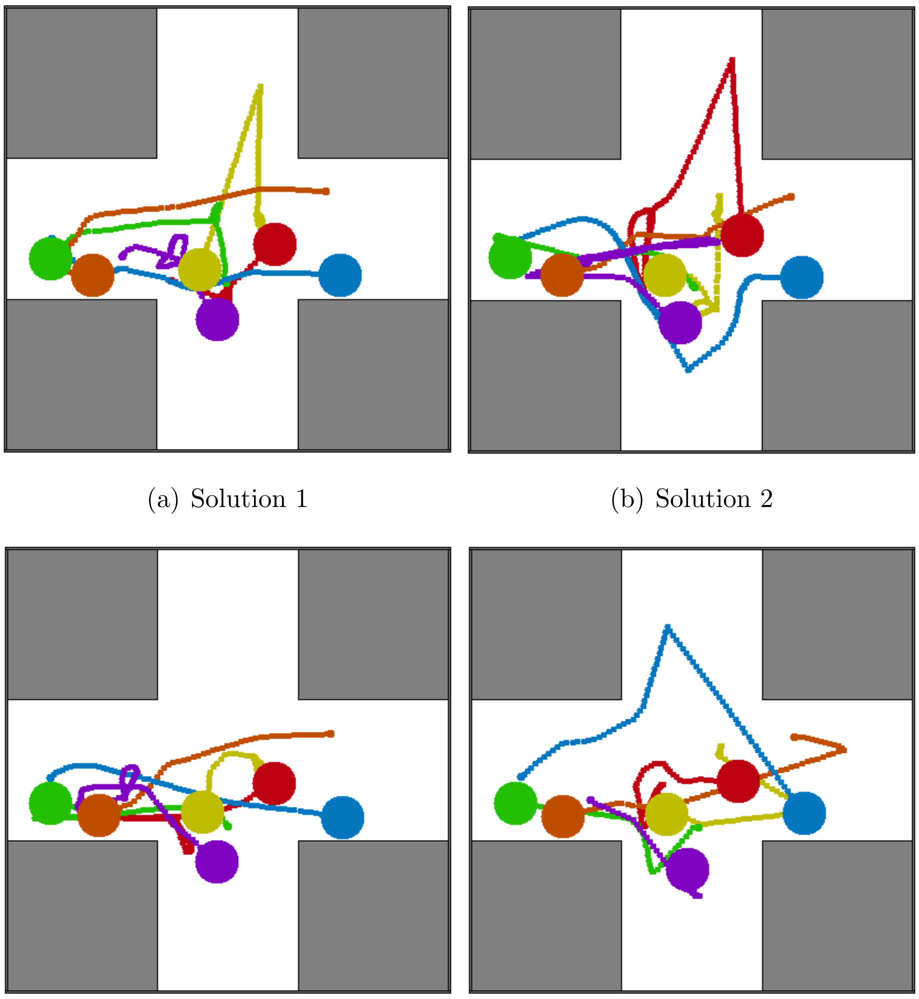
6.4.3 Ranking of the algorithms by their suboptimality

Figure 6.11 and 6.12 show the histograms of ranks assigned to algorithms for run-time limits 5 and 1 seconds. A rank from 1 to 4 is assigned to each algorithm for each experiment according to its solution suboptimality compared to other algorithms. If two algorithms achieve the same suboptimality, the ranks are assigned to them randomly. If an algorithm was not able to find any solution, its rank is 4. Rank 1 means that an algorithm achieved lowest suboptimality for particular problem instance. Difference between figures 6.11 and 6.12 shows how the ranks of the algorithms (e.g. solution quality – suboptimality) depend on the running time limit.

The results show that the VisibilityGraph-RRT* algorithm achieved the worst ranks. Better ranks were achieved by the Line-RRT* algorithm due to its ability of fast sampling of the state space. The ORCA algorithm shares the first rank with the ORCA-RRT* for the 1 second runtime limit, where the ORCA-RRT* cannot benefit from extra computation time. When the 5 second runtime limit is considered, the ORCA-RRT* algorithm is dominant in terms of lower suboptimality.

From the difference between results for different runtime limits (see Figures 6.11 and 6.12) we can observe that ORCA finds a solution early, but does not benefit from the remaining runtime limit. On the other hand, the performance of the RRT* based algorithms including the ORCA-RRT* is more dependent on the runtime limit.

Altogether, the results in Figures 6.11 and 6.12 confirm that ORCA-RRT* has not only the best α-coverage of the problem instance space, but also is dominant in terms of better suboptimality of the solutions, which implies their better quality.

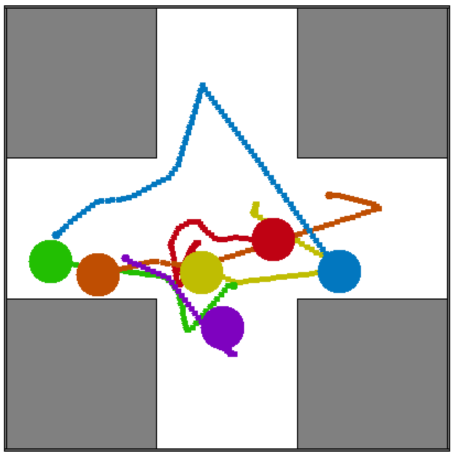

(a) Solution 1 (b) Solution 2

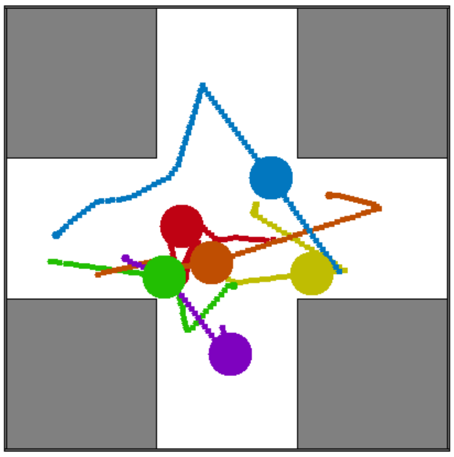
(c) Solution 3 (d) Solution 4

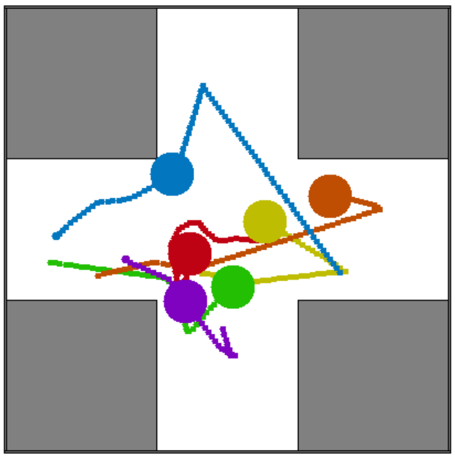
Figure 6.3: Solutions found by ORCA-RRT*, running time: 5 seconds, suboptimality threshold α: 10, number of agents n: 6, radius of agents' body r: 50

Solution	Time [ms]	Iteration	Suboptimality
1	1629	18	10.0
2	1791	20	5.2
3	2173	23	3.8
4	3902	54	3.0

Table 6.1: Solutions found by ORCA-RRT*, running time: 5 seconds, suboptimality threshold α: 10, number of agents n: 6, radius of agents' body r: 50


(a) Time 1


(b) Time 21


(c) Time 70

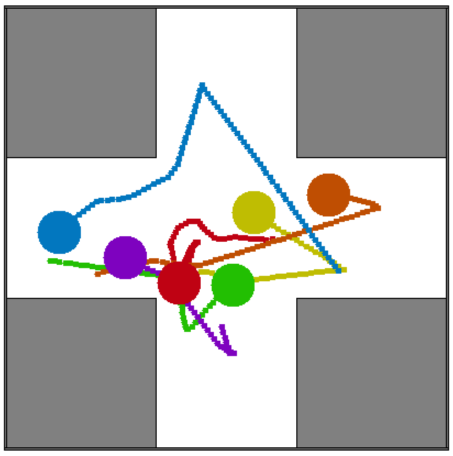
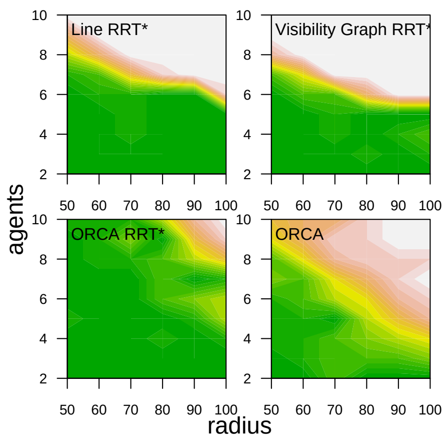
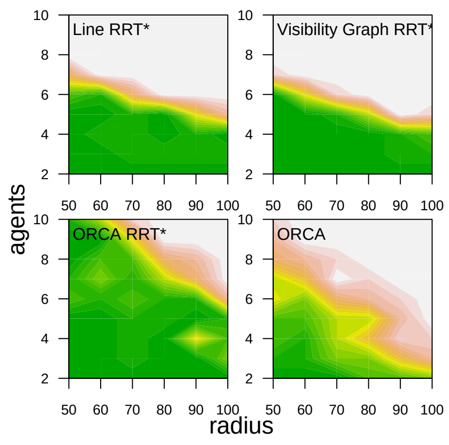

(d) Time 94

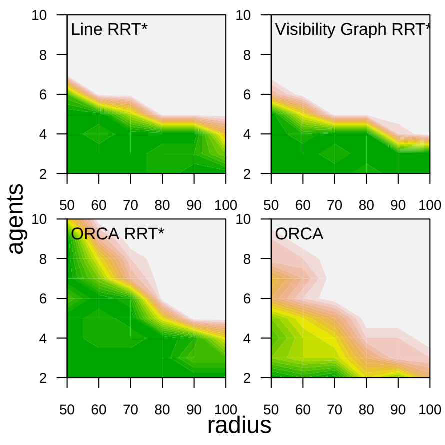
Figure 6.4: Simulated execution of the solution found by ORCA-RRT*, running time: 5 seconds, suboptimality threshold α: 10, number of agents n: 6, radius of agents' body r: 50



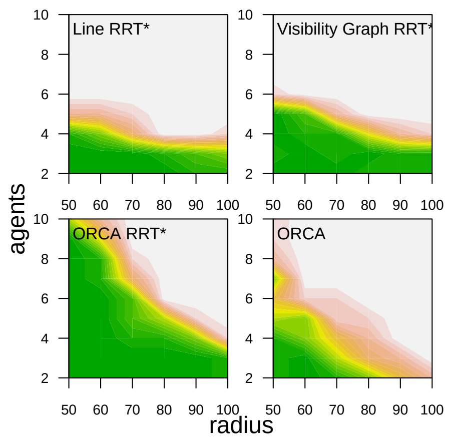
(a) Empty environment



(b) Door environment



(c) Cross environment



(d) Maze environment

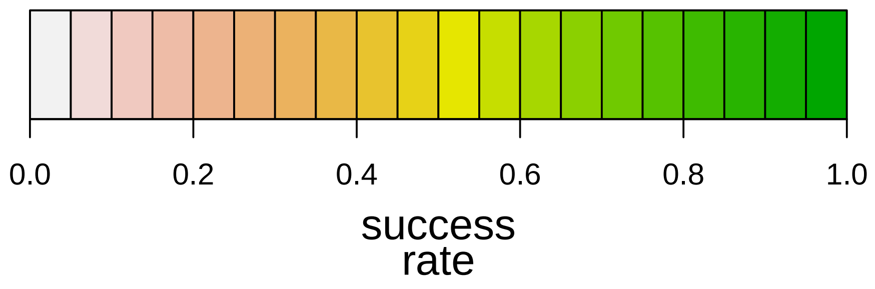
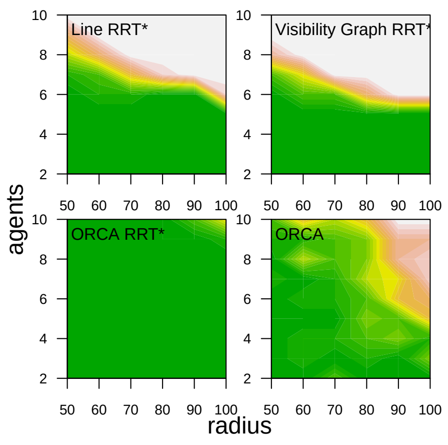
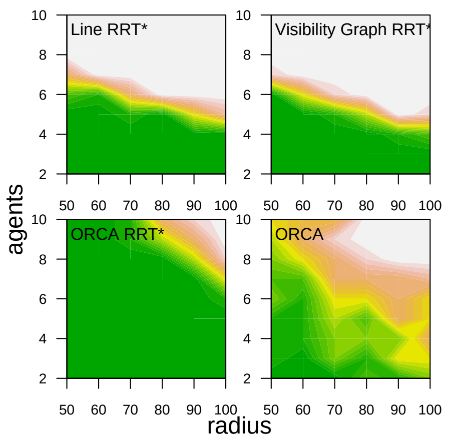


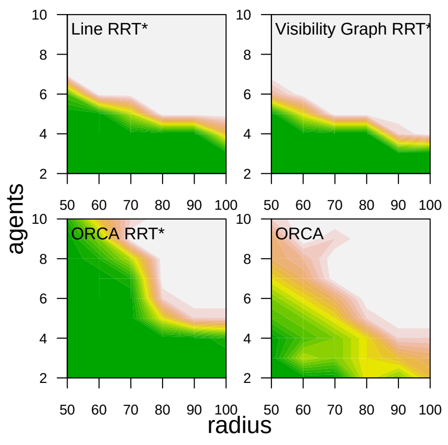
Figure 6.5: Success rate of tested algorithms on test instances, running time: 5 seconds, suboptimality threshold α: 2.5



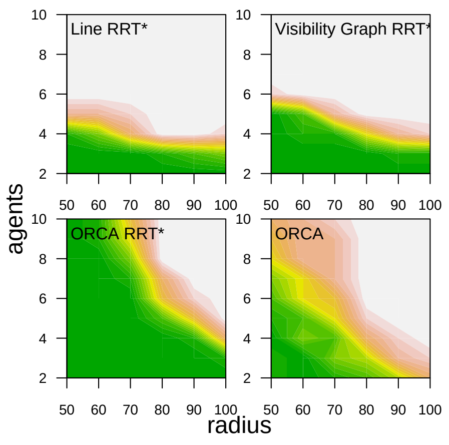
(a) Empty environment



(b) Door environment



(c) Cross environment



(d) Maze environment

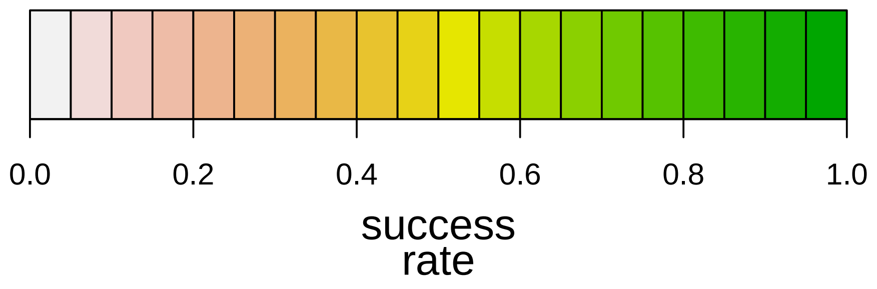
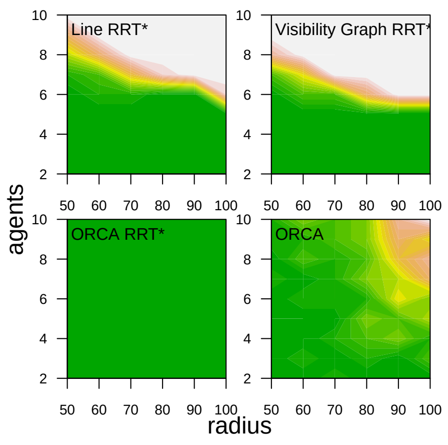
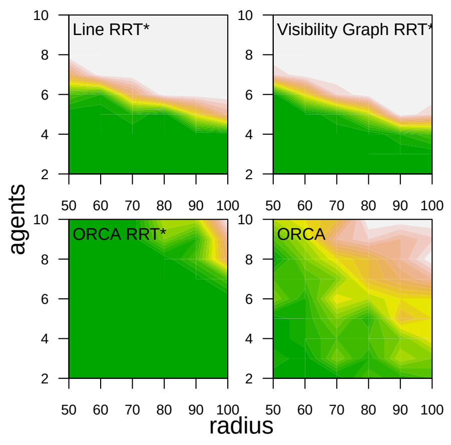


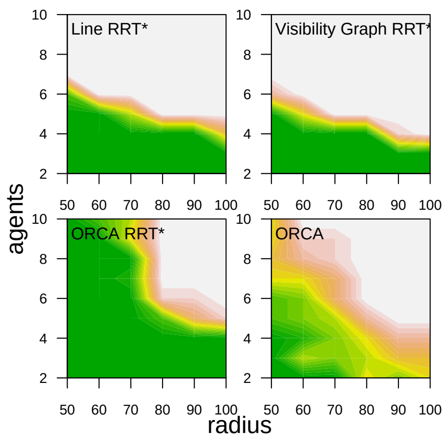
Figure 6.6: Success rate of tested algorithms on test instances, running time: 5 seconds, suboptimality threshold α: 5



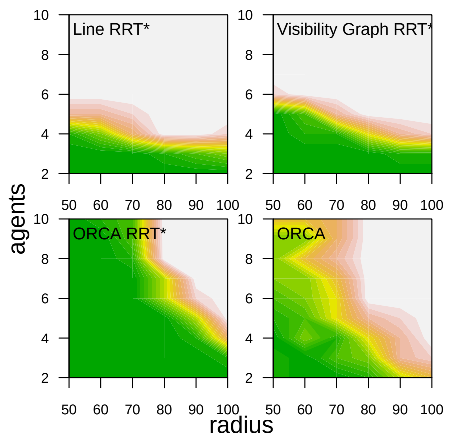
(a) Empty environment



(b) Door environment



(c) Cross environment



(d) Maze environment

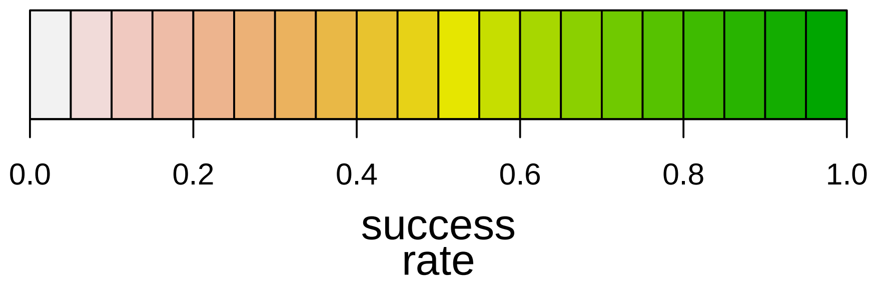
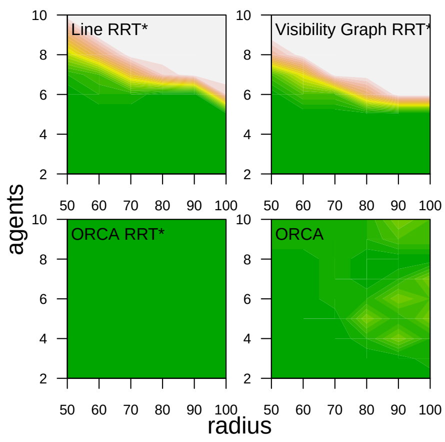
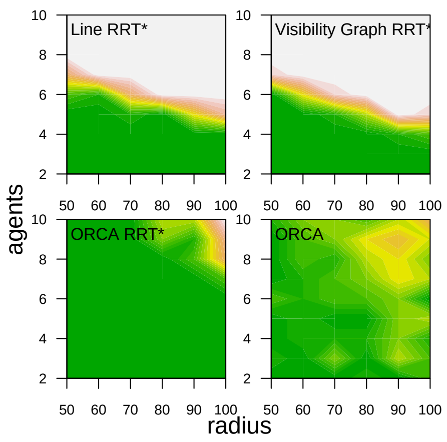


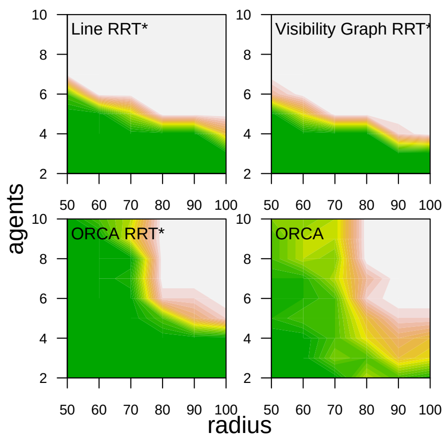
Figure 6.7: Success rate of tested algorithms on test instances, running time: 5 seconds, suboptimality threshold α: 10



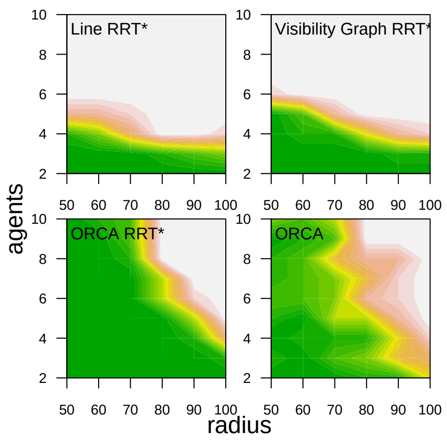
(a) Empty environment



(b) Door environment



(c) Cross environment



(d) Maze environment

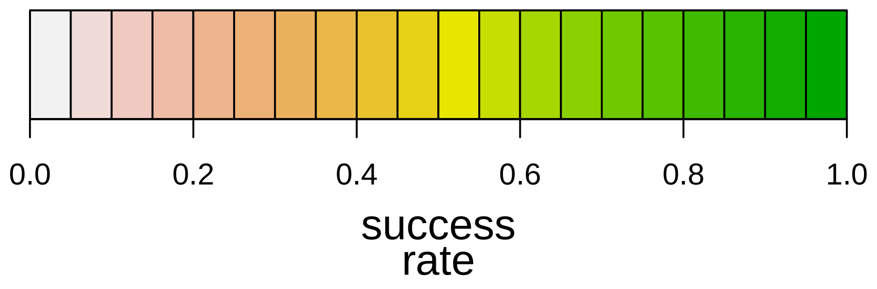


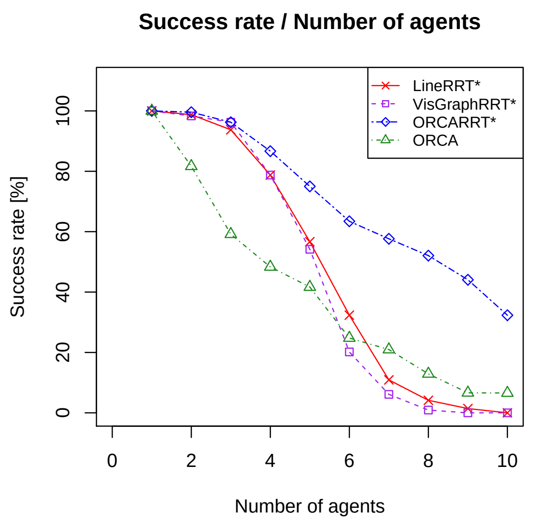
Figure 6.8: Success rate of tested algorithms on test instances, running time: 5 seconds, no suboptimality threshold α

Number of agents	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Line-RRT*	100	98.8	93.8	78.8	56.7	32.4	10.9	4.2	1.4	0
Vis.Graph-RRT*	100	98.3	96.3	78.8	54.2	20.2	6.1	0.9	0	0
ORCA	100	81.7	59.2	48.3	41.7	24.8	21	12.9	6.6	6.6
ORCA-RRT*	100	99.6	96.3	86.7	75	63.5	57.6	52.1	44.1	32.3

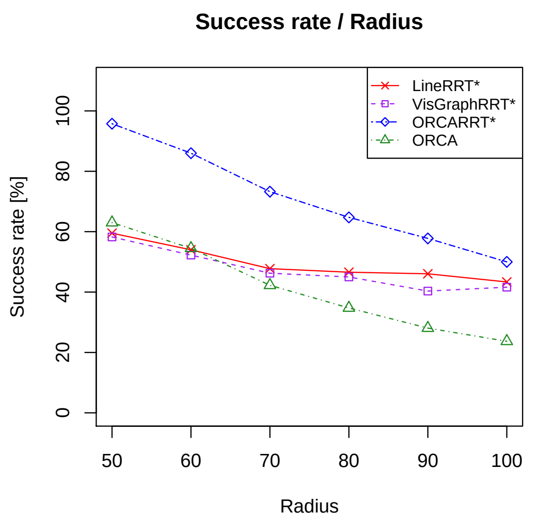
Table 6.2: Success rate of the measured algorithms for various numbers of agents, running time: 5 seconds, suboptimality threshold α: 2.5

Radius of agents' bodies	**50**	**60**	**70**	**80**	**90**	**100**
Line-RRT*	59.5	54	47.8	46.6	46.1	43.4
VisibilityGraph-RRT*	58.3	52.3	46.3	45	40.3	41.6
ORCA	63	54.5	42.3	34.7	28.1	23.7
ORCA-RRT*	95.8	86	73.3	64.7	57.8	50

Table 6.3: Success rate of the measured algorithms for various radii of agents' bodies, running time: 5 seconds, suboptimality threshold α: 2.5



(a) Success rate by the number of agents



(b) Success rate by the radius of agents

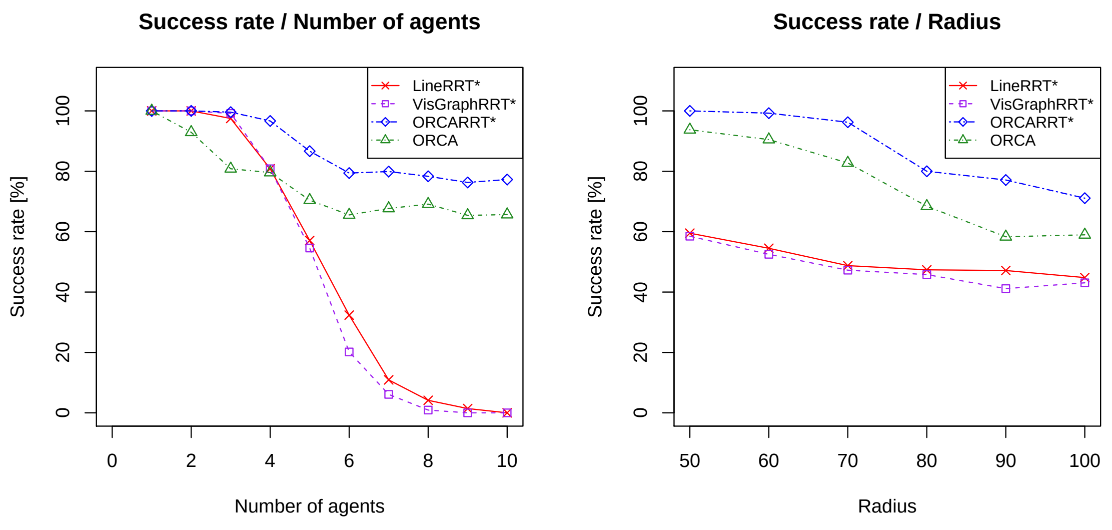
Figure 6.9: Success rate of the measured algorithms for various numbers of agents and radii of agents' bodies, running time: 5 seconds, suboptimality threshold α: 2.5

Number of agents	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Line-RRT*	100	100	97.5	80.8	57.1	32.4	10.9	4.2	1.4	0
Vis.Graph-RRT*	100	100	99.2	80.8	54.6	20.2	6.1	0.9	0	0
ORCA	100	92.9	80.8	79.6	70.4	65.5	67.7	69.1	65.4	65.7
ORCA-RRT*	100	100	99.6	96.7	86.7	79.4	79.9	78.3	76.3	77.3

Table 6.4: Success rate of the measured algorithms for various numbers of agents, running time: 5 seconds, no suboptimality threshold α

Radius of agents' bodies	**50**	**60**	**70**	**80**	**90**	**100**
Line-RRT*	59.5	54.5	48.8	47.4	47.1	44.8
VisibilityGraph-RRT*	58.5	52.5	47.3	45.8	41.1	43.1
ORCA	93.8	90.5	82.8	68.4	58.3	59
ORCA-RRT*	100	99.3	96.3	80	77.1	71.1

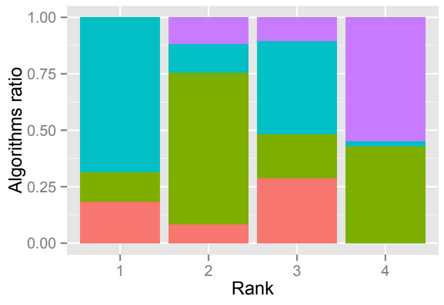
Table 6.5: Success rate of the measured algorithms for various radii of agents' bodies, running time: 5 seconds, no suboptimality threshold α



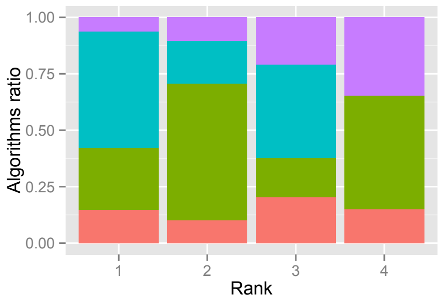
(a) Success rate by the number of agents

(b) Success rate by the radius of agents

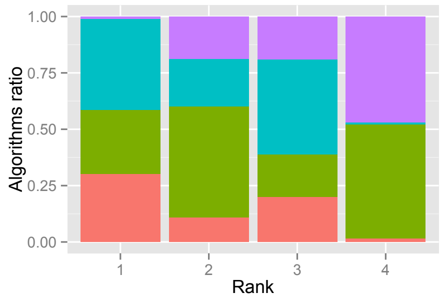
Figure 6.10: Success rate of the measured algorithms for various numbers of agents and radii of agents' bodies, running time: 5 seconds, no suboptimality threshold α



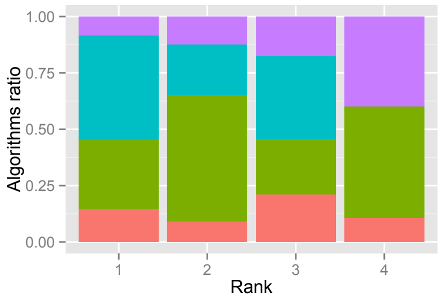
(a) Empty environment



(b) Door environment



(c) Cross environment



(d) Maze environment

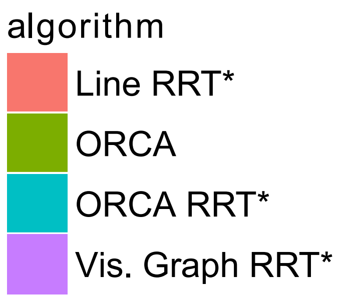
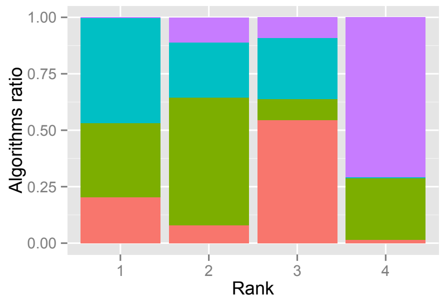
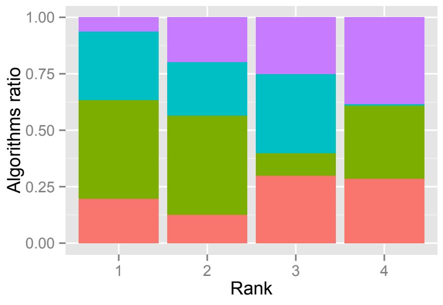


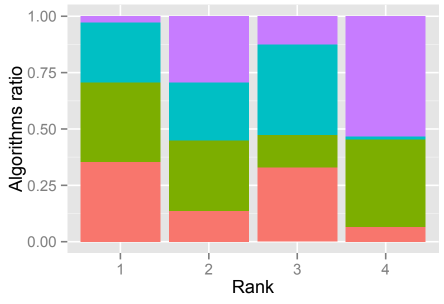
Figure 6.11: Rank histograms, running time: 5 seconds, suboptimality threshold α: 2.5



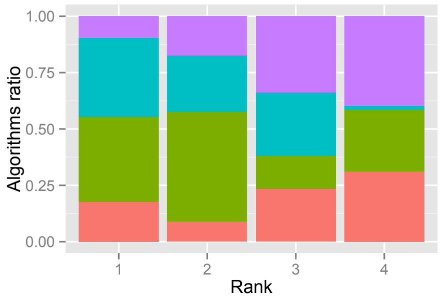
(a) Empty environment



(b) Door environment



(c) Cross environment



(d) Maze environment

algorithm

🟥 Line RRT*

🟩 ORCA

🟦 ORCA RRT*

🟪 Vis. Graph RRT*

Figure 6.12: Rank histograms, running time: 1 second, suboptimality threshold α: 2.5

Chapter 7

Conclusion

In this thesis we studied the problem of finding coordinated paths for holonomic agents in 2-d polygonal environments. This problem is challenging because it belongs to PSPACE-hard and complete algorithms have to search for a solution in a state space exponential to the number of agents. In this thesis we study the well known techniques that are able to solve this problem - reactive collision avoidance techniques and a sampling based planning approach. We provide an overview of the reactive techniques for collision avoidance, focusing on a well known velocity obstacle approach. Also we study a sampling based planning approach - the RRT* algorithm.

We studied a reactive technique ORCA and several RRT*-based algorithms for multi-agent path coordination. We found typical instances of the multi-agent path coordination problem that the ORCA reactive technique fails to solve. While both ORCA and RRT* have limited coverage of the problem instance space, an approach combining planning and reactive technique benefits from both its parts, providing a better problem instance set coverage along with higher solution quality.

We call the new algorithm ORCA-RRT*. While reactive techniques are often unable to solve problems containing local minima, due to its RRT* planning part the ORCA-RRT* algorithm can avoid such local minima by random sampling of the state space. On the other hand RRT*-based algorithms often suffer from the exponential growth of the state space and thus are unable to solve instances with high number of agents. The reactive part of ORCA-RRT* is able to overcome this problem.

We proved that, with respect to a given upper bound on the acceptable suboptimality of a solution, the coverage of ORCA-RRT* is a superset of the coverage of ORCA i.e. there is no instance that would be solved by ORCA and not solved by ORCA-RRT*, while there are instances solved by ORCA-RRT* only. Our experiment results confirm this property of the algorithm.

ORCA-RRT* is an anytime algorithm, which can iteratively improve the solution it provides. The choice of a running time limit can therefore significantly affect its performance. We experimented with several running time limits and found that the performance of the RRT* based algorithms including ORCA-RRT* is more dependent on the running time limit than the performance of the ORCA reactive technique.

In the future work we plan to implement the ORCA-RRT* algorithm on hardware agents and run tests in real environments. Another interesting research aim is an ORCA-RRT* based algorithm capable of finding coordinated paths for non-holonomic agents.

Bibliography

Alonso-Mora, J., Breitenmoser, A., Beardsley, P. and Siegwart, R. (2012), 'Reciprocal collision avoidance for multiple car-like robots', *Proceedings of the 2012 IEEE International Conference on Robotics and Automation ICRA* pp. 360–366.

Bentley, J. L. (1975), 'Multidimensional binary search trees used for associative searching', *Communications of the ACM* **18**, 509–517.

Cap, M., Novak, P., Jiri, V. and Michal, P. (2013), 'Multi-agent RRT *: Sampling-based cooperative pathfinding', *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems* pp. 1263–1264.

de Wilde, B., ter Mors, A. W. and Witteveen, C. (2013), 'Push and rotate: cooperative multi-agent path planning', *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems* pp. 87–94.

Erdmann, M. and Lozano-Perez, T. (1986), 'On multiple moving objects', *Robotics and Automation. Proceedings. 1986 IEEE International Conference* **3**, 1419 – 1424.

Fiorini, P. and Shiller, Z. (1998), 'Motion Planning in Dynamic Environments Using Velocity Obstacles', *The International Journal of Robotics Research* **17**(7), 760–772.

Fredman, M. L. and Tarjan, R. E. (1984), 'Fibonacci heaps and their uses in improved network optimization algorithms', *25th Annual Symposium on Foundations of Computer Science. IEEE.* pp. 338–346.

Guy, S. J., Chhugani, J., Kim, C., Satish, N., Lin, M., Manocha, D. and Dubey, P. (2009), 'ClearPath : Highly Parallel Collision Avoidance for Multi-Agent Simulation', *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* pp. 177–187.

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

Karaman and Frazzoli (2011), ‘Sampling-based algorithms for optimal motion planning’, *The International Journal of Robotics Research* **30**, 846–894.

Lalish, E. and Morgansen, K. A. (2009), ‘Distributed reactive collision avoidance’, *Autonomous Robots* **32**(3), 207–226.

LaValle, S. M. (2004), *Planning Algorithms*, Cambridge University Press.

LaValle, S. M. and Kuffner, J. J. (1999), ‘Randomized kinodynamic planning’, *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference* **1**, 473–479.

Miller, C. (2012). http://www.mhi.org/media/news/11336.

Otte, M. and Correll, N. (2010), ‘Any-com multi-robot path-planning with dynamic teams: Multi-robot coordination under communication constraints’, *Experimental Robotics, The 12th International Symposium on Experimental Robotics* pp. 743–757.

Reif, J. and Sharir, M. (1985), ‘Motion planning in the presence of moving obstacles’, *Foundations of Computer Science, 1985., 26th Annual Symposium* pp. 144–154.

Russell, S. J. and Norvig, P. (2009), *Artificial Intelligence: A Modern Approach*, Prentice Hall.

Surynek (2009), ‘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* pp. 3613–3619.

van den Berg, J., Guy, S. J., Lin, M. and Manocha, D. (2011), ‘Reciprocal n-body collision avoidance’, *Robotics Research: The 14th International Symposium ISRR, Springer Tracts in Advanced Robotics, vol. 70, Springer-Verlag, May 2011* pp. 3–19.

van den Berg, J., Guy, S. J., Snape, J., Lin, M. and Manocha, D. (2012), ‘Rvo2 library: Reciprocal collision avoidance for real-time multi-agent simulation’. http://gamma.cs.unc.edu/RVO2/.

van den Berg, J. and Manocha, D. (2008), 'Reciprocal Velocity Obstacles for real-time multi-agent navigation', *2008 IEEE International Conference on Robotics and Automation* pp. 1928–1935.

Appendix A

Source codes

A *java* implementation of the ORCA-RRT* algorithm along with the ORCA, Line-RRT* and VisibilityGraph-RRT* algorithms is enclosed on the CD in the following directory:

- implementation/ORCA-RRT: The implemented ORCA-RRT project containing all of the mentioned algorithms and the benchmark set generator along with the generated set of problem instances
- implementation/dependencies: dependencies required to run the ORCA-RRT project in eclipse. Projects alite, trajectorytools and deconflictiontools were developed at the Agent Technology Center, Czech Technical University in Prague
- implementation/export: The ORCA-RRT project exported in a runnable jar file

Please use the following arguments to run the project:

- In case of running the project in eclipse:
  <instance-number> <problem-instance.xml> <algorithm> <random-seed> <time-limit-in-ms> <suboptimality-threshold> <true/false-show-visualization> <eclipse>
- In case of running the exported version from the command line:
  <instance-number> <problem-instance.xml> <algorithm> <random-seed> <time-limit-in-ms> <suboptimality-threshold> <true/false-show-visualization>

The instance number is arbitrary, it was used for tracking experiments, the problem instance has to be an xml file with the same structure as the provided problem instance files (ORCA-RRT/src/main/resources/instances), for algorithm choose from: ORCA-RRT, ORCA, LINERRT, VGRRT.

Appendix B

Paper for AAMAS 2014

Finding Coordinated Paths for Multiple Holonomic Agents in 2-d Polygonal Environment

During the work on this thesis a paper on the ORCA-RRT* algorithm has been submitted by Bc. Pavel Janovský, Bc. Michal Čáp MSc. and Ing. Jiří Vokřínek Ph.D. to the 13th International Conference on Autonomous Agents and Multiagent Systems AAMAS 2014. The author notification date was after the submission of this thesis.

We include the paper on the enclosed CD:

- paper/ORCA-RRT-paper.pdf