FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Multi-agent Path Finding in Continuous Environment
Student:	Bc. Kristýna Janovská
Supervisor:	prof. RNDr. Pavel Surynek, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2024/2025

Instructions

The aim of the work is to propose techniques for multi-agent pathfinding (MAPF) in a continuous environment, where the task is to find a collision-free movement plan for a group of agents. An obstacle to an agent can be both a static object and another agent. As an innovation, we want to allow agents to move along smooth curves. Current approaches for MAPF that consider continuity only consider continuous time and not environment [1]. As objective functions, the duration of the plan can be considered, but also, for example, the visual quality of the plan (smooth movement of agents).

The tasks for the student are as follows:

1. Study algorithms for multi-agent pathfinding considering continuity, then focus on single-agent pathfinding algorithms in a continuous space [3, 4].

2. Design an algorithm or a modification of an existing algorithm for multi-agent pathfinding that will consider a continuous environment. For example, a modification of the CCBS algorithm [1] can be tried, where the collision between agents would be eliminated not by waiting, but by an evasive maneuver along a smooth curve.

3. Implement your design in the form of a software prototype and verify experimentally in relevant test scenarios.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 20 December 2023 in Prague.

[1] Anton Andreychuk, Konstantin S. Yakovlev, Pavel Surynek, Dor Atzmon, Roni Stern: Multi-agent pathfinding with continuous time. Artif. Intell. 305: 103662 (2022)
[2] Cao N, Yi G, Zhang S, Qiu L. A multiobjective path-smoothing algorithm based on node adjustment and turn-smoothing. Measurement and Control. 2023;56(7-8):1187-1201. doi: 10.1177/00202940221139327
[3] Karaman, S.; Frazzoli, E. Sampling-based Algorithms for Optimal Motion Planning. CoRR, volume abs/1105.1186, 2011, 1105.1186. Available from: http://arxiv.org/abs/1105.1186
[4] Garrido Carpio, Fernando. (2018). Two-staged local trajectory planning based on optimal pre-planned curves interpolation for human-like driving in urban areas.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master’s thesis

Multi-agent Path Finding in Continuous Environment

Bc. Kristýna Janovská

Department of applied mathematics
Supervisor: prof. RNDr. Pavel Surynek, Ph.D.

May 9, 2024

Acknowledgements

I would like to thank my family, friends, and my supervisor prof. RNDr. Pavel Surynek, Ph.D., for support during the writing of this thesis. I also thank apes for making it possible for us humans to evolve and Darwin for figuring it out.

Declaration

In Prague on May 9, 2024

Citation of this thesis

Janovská, Kristýna. *Multi-agent Path Finding in Continuous Environment.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Abstrakt

Tato práce je zaměřena na vývoj modelu pro bezkolizní multi-agentní hledání cest ve spojitém prostoru, kde se agenti pohybují po množinách hladkých křivek. Agenti mezi sebou komunikují za použití algoritmu Continuous Environment Conflict-Based Search (CE-CBS), který je v této práci navržen.

Jsou zde studovány různé parametry modelu a typy prostředí a nastavení agentů, aby bylo zjištěno, jak se agenti chovají v závislosti na náročnosti prostoru a jak se mění kvalita řešení. Výsledky těchto experimentů ukazují schopnost agentů pohybovat se po bezkolizních hladkých cestách, kdy zároveň optimalizují délky těchto cest.

Klíčová slova MAPF, spojitý prostor, CCBS, CE-CBS, B-spline křivky, RRT*

Abstract

This work sets to develop a model of collision-free multi-agent path finding in continuous environment, where agents move on sets of smooth curves. Agents communicate between themselves using the Continuous Environment Conflict-Based Search (CE-CBS) algorithm proposed in this work.

Various parameters of this model and different types of environment and agent setting are studied to determine how agents behave based on the difficulty of the environment setting and how the solution quality changes. The results of these experiments show the ability of agents to move on non-colliding smooth paths while also optimising the path costs.

Keywords MAPF, continuous environment, CCBS, CE-CBS, B-spline curves, RRT*

Contents

List of Figures

List of Tables

Introduction

In recent years, the problem of path planning has changed from movement on a discrete graph to movement in a continuous space. The multi-agent path finding problem, in which several agents, or robots, move at once in an environment without colliding. For a discrete problem, this means that two agents are not present in the same vertex at the same time step. However, in a continuous problem that aims to realistically solve real-life scenarios such as warehouse logistics or even autonomous vehicle movement, both time and space are continuous. The agents themselves should also be represented by a rigid body. Collisions between agents should therefore mean that two agents overlap at a given time. It is also possible to take various kinematic and dynamic constraints into account which constrain the agent's movement.

The goal of this work is to create a software prototype that allows continuous movement in both time and space for multiple agents at once. Agents will move on a set of smooth curves that will create their paths. These paths will be constrained so that no agents collide at the same time. As agents are rigid circular bodies in this work, a collision will occur if two agents overlap at any time. An algorithm based on existing Continuous Conflict-Based Search [Andreychuk et al., 2022] will be proposed. This algorithm works in continuous time, but the proposed modification will also work in a continuous environment. Unlike CCBS, agents will plan their paths using the RRT* algorithm [Karaman and Frazzoli, 2011], whose output will then be smoothed with B-spline curves. Single-agent methods for smooth path planning have already been studied. This work will build on them to provide a non-conflicting smooth multi-agent solution.

The work is sectioned as follows. Firstly, the theoretical background of this problem will be studied. This will contain the problem of continuous multi-agent path finding, the most important algorithms such as CCBS [Andreychuk et al., 2022] or RRT*[Karaman and Frazzoli, 2011], and smoothing with B-spline curves, as well as selected related works to all of these topics will be discussed. Next, the proposed model consisting of several base parts will be explained. These parts will include a constrained single-agent path finding algorithm, a smoothing mechanism, and a multi-agent algorithm for path finding in a continuous environment called CE-CBS. In the last part, performed experiments are discussed, with several important parameters being introduced and their impact on the model's performance being studied.

Chapter 1

Background

1.1 Multi-agent Path Finding

This work focuses on guiding a set of artificial agents, entities that are capable of observing their environment through sensors and acting based on this input through actuators. Actions which agents are to take are described by their agent functions, which maps percept sequences - input of sensors - to actions. An agent function is implemented by an agent program, which together with an architecture completes an agent. Architecture saves information about the environment around an agent [Russell and Norvig, 2020].

In this work, agent-based modelling (ABM) [Bonabeau, 2002] is used to describe a collection of agents. Each of these agents makes their own decisions, although they are controlled centrally and communicate and take other agents into account. The agents used here are goal-based agents with prior knowledge of the environment [Russell and Norvig, 2020].

1.1.1 Discrete MAPF

Definition 1.1.1 (Multi-agent Path Finding)**.** Let G = (V, E) be an undirected graph and let $R = \{r_1, r_2, ..., r_v\}$ be a set of agents, $v < |V|$.

The initial arrangement of agents is defined by a simple function $S_R^0 : R \to V$, $S_R^0(r) \neq S_R^0(s)$ for each $r, s \in R, r \neq s$.

The goal arrangement of agents is a simple function $S_R^+ : R \to V$, $S_R^+(r) \neq S_R^+(s)$ for each $r, s \in R, r \neq s$.

A problem of multi-agent path finding is the task of finding a number ζ and a sequence $\mathcal{S}_R = [S_R^0, S_R^1, ..., S_R^\zeta]$, where $S_R^k : R \to V$ is a simple function for each $k = 1, 2, ..., \zeta$. This sequence must satisfy the following constraints:

(i) $S_R^\zeta = S_R^+$; meaning all agents reach their goal vertices.

(ii) Let $S_R^k(r) = S_R^{k+1}(r)$, or $\{S_R^k(r), S_R^{k+1}(r)\} \in E$ for each $r \in R$ a $k = 1, 2, ..., \zeta - 1$; that is, an agent may stay at its vertex or move to a neighbouring vertex between two time steps.

(iii) If $S_R^k(r) \neq S_R^{k+1}(r)$ (agent r makes a move between time steps k and $k+1$) and $S_R^k(s) \neq S_R^{k+1}(r)$ $\forall s \in R$ such that $s \neq r$ (that is, no other agent s is present in the goal vertex in time step k), then movement of agent

r in time step k k is called allowed (agent r moves into an unoccupied neighbouring vertex).

If $S_R^k(r) \neq S_R^{k+1}(r)$ and it exists $s \in R$ such that $s \neq r \wedge S_R^k(s) = S_R^{k+1}(r) \wedge S_R^k(s) \neq S_R^{k+1}(s)$ (agent r is moving into a vertex, which is currently being freed by an agent s) and the movement of agent s in time step k is allowed, then the movement r in time step k is also allowed.

All the movements of agents in all the time steps must be allowed. Analogically, this condition along with the simplicity condition of functions consisting of S_R implies that no two agents can enter the same vertex in the same time step.

The instance of the MAPF problem is formally a quadruple $\Sigma = (G, R, S_R^0, S_R^+)$. The solution Σ can be noted as $\mathcal{S}_R(\Sigma) = [S_R^0, S_R^1, ..., S_R^\zeta]$. [Surynek, 2010]

1.1.2 Continuous Time MAPF

The problem of continuity in multi-agent path finding can be addressed from several viewpoints – with continuous representation of time, or a continuous representation of environment. In discrete problems, time would be represented as discretised time steps, and the environment where agents are present is represented as a graph in whose vertices the agents are situated.

$MAPF_R$ is a version of the MAPF problem with continuous time. Algorithms that address this problem are, for example, $CCBS, E-ICTS$ or $ECBS-CT$ [Andreychuk et al., 2022, Walker et al., 2018, Cohen et al., 2021]. Of these, only $CCBS$ provides a solution without the need for any discretization of time.

Definition 1.1.2 (MAPF$_R$)**.** is the tuple $\mathcal{E} = [\mathcal{G} = (V, E), \mathcal{M}, S, G, coord, \mathcal{A}]$, where $\mathcal{G}$ represents a graph, $\mathcal{M}$ a metric space, S the start function, and G the goal function with $coord$ mapping every vertex in $\mathcal{G}$ to a coordinate in $\mathcal{M}$ and $\mathcal{A}$ being a finite set of possible move actions.

Every action $a \in \mathcal{A}$ in MAPF$_R$ is defined by a duration a_D and a motion function $a_\varphi : [0, a_D] :\to \mathcal{M}$ which maps time to metric space. $a_\varphi(t)$ is the coordinate of an agent in the metric space $\mathcal{M}$ at a time t while executing an action a. [Andreychuk et al., 2022]

This definition allows for non-constant speed of movement of agents, which can also differ from agent to agent.

Definition 1.1.3 (Conflict in MAPF$_R$)**.** Two single agent plans π_i and π_j have a conflict if $\exists t \in [0, max(\pi_i D, \pi_j D)] : IsCollision(i, j, \pi_i\varphi(t), \pi_i\varphi(t))$. [Andreychuk et al., 2022]

1.1.2.1 Smooth Continuous MAPF - SC-MAPF

In this work, both continuous environment and time are assumed. The agents are to find non-conflicting paths satisfying several properties. The paths are defined as smooth curves and have to satisfy certain kinematic constraints. Here, that constraint is the minimal angle constraint, setting a minimal value to angles between path segments before smoothing takes place. Agent's paths can cross, but the agents themselves cannot collide - they cannot overlap at any time. The movement of agents on their paths in time is represented by their trajectories, which are checked for conflicts. As in standard MAPF, agents start in pre-set starting position and they try to reach their goal positions.

The following definition is based on existing definitions by [Surynek, 2010, Čáp et al., 2015, Neto et al., 2010, Čáp et al., 2013, Andreychuk et al., 2022].

Definition 1.1.4 (Smooth Continuous MAPF - SC-MAPF)**.** Let $\mathcal{M}$ be a metric space, and $R = \{r_1, r_2, ..., r_v\}$ a set of agents.

SC-MAPF is the tuple $\mathcal{E} = [\mathcal{G} = (V, E), \mathcal{M}, S, G, coord, \mathcal{A}]$, where $\mathcal{G}$ represents a graph, $\mathcal{M}$ a metric space, S the start function and G the goal function with *coord* mapping every vertex in $\mathcal{G}$ to a coordinate in $\mathcal{M}$ and $\mathcal{A}$ being a finite set of possible move actions. Every edge E in $\mathcal{G}$ represents a set of smooth curves that connect two vertices in $\mathcal{G}$. A set of curves belonging to an edge $e \in E$ is denoted as c_e.

Every action $a \in \mathcal{A}$ in MAPF$_R$ is defined by a duration a_D and a motion function $a_\varphi : [0, a_D] :\rightarrow \mathcal{M}$ that maps time to metric space. This motion function corresponds to a smooth curve. $a_\varphi(t)$ is the coordinate of an agent in the metric space $\mathcal{M}$ at a time t while executing an action a.

The agent body is defined as a shape in metric space.

The initial configuration of agents is defined by a simple function $S_R : R \rightarrow V$, $S_R(r) \neq S_R(s)$ for each $r, s \in R, r \neq s$.

The goal configuration of agents is a simple function $G_R : R \rightarrow V$, $G_R(r) \neq G_R(s)$ for each $r, s \in R, r \neq s$.

A problem of smooth continuous multi-agent path finding is a task of finding a set of smooth curves - a sequence of actions, each choosing a curve c from a set of curves c_e belonging to a respective edge e, for each agent so that no agents collide.

A conflict is defined as a 5-tuple $\mathcal{C} = [\alpha_i, a(\alpha_i), \alpha_j, a(\alpha_j), t]$, where α is an agent, $a(\alpha)$ is the action agent α performs at the time the conflict occurs and t is the beginning of the time interval when the conflict occurs. This means that two agents overlap. Solving a conflict means forbidding the agents to move on certain curves which would traverse through the conflicting position at a conflicting time.

The solution of this problem is a set of actions of a set of agents satisfying the following constraints:

(i) $S_R = G_R$; meaning all agents reach their goal positions.

(ii) An agent moves at a set speed.

(iii) An agent's trajectory is a smooth curve, which satisfies given kinematic constraints.

(iv) A sequence of actions can be valid only if no two agent bodies overlap at any time.

In this work, agent bodies are defined as circles with fixed radius.

Algorithm 1: Higher level of CBS [Sharon et al., 2014]

```
Input: MAPF instance
1  R.constraints = $\emptyset$;
2  R.solution = set of individual agent paths;
3  R.cost $\leftarrow$ compute_cost(R.solution);
4  OPEN $\leftarrow$ R;
5  while OPEN not empty do
6      P $\leftarrow$ lowest_cost_node(OPEN);
7      $first\_conflict$ $\leftarrow$ validate(P);
8      if first_conflict = $\emptyset$ then
9          return P.solution;
10     for $agent_i$ in first_conflict do
11         N $\leftarrow$ $new\_node$;
12         N.constraints $\leftarrow$ P.constraints $\cup$ $new\_constraint(agent_i$,
             vertex, time);
13         N.solution $\leftarrow$ P.solution;
14         N.solution.update($new\_constraint$());
15         N.cost $\leftarrow$ compute_cost(N);
16         Insert N to OPEN;
```

1.2 Discrete CBS algorithm

The CBS algorithm is a cost-optimal MAPF algorithm. The goal of this algorithm is to find a set of paths for a set of agents, so that no agents collide at any step of their paths, if a solution exists, since it is a solution-complete algorithm [Andreychuk et al., 2022]. The objective is to find a conflict-free plan that minimises the makespan, which is the maximum of agents' path costs. CBS is a discrete algorithm; the environment is represented by a graph, and time is discretised into time steps. In every time step, an agent may perform one action, either a movement into a neighbouring vertex or a waiting action, that is, staying in its current vertex.

This algorithm consists of two levels; the lower level computes a path for each agent with the help of a single-agent path finding algorithm such as A*. This path must satisfy the constraints provided by the higher level of CBS.

A constraint arises from collision solving, which takes place at the higher level. A constraint includes an agent a, a vertex v, and a time step t, which means that the specific agent a cannot be present in v at step t.

The higher level of CBS first computes paths for all agents and then checks each step of these paths for collisions. Collision is a situation where two or more agents are present in a vertex at a time.

CBS searches for a conflict-free solution by constructing and searching a constraint tree, a binary tree where each node contains a set of constraints, a solution, and the total cost of that solution. Two new nodes are created when a node is searched and a conflict is found within its solution. A conflict is between two agents, so it can be resolved in two ways - either one agent is constrained from entering that vertex at a given time, or the second agent. Therefore, two new possible sets of constraints are created and both have to be verified. New paths containing one of the new constraints are found, and two new nodes are created.

If a solution of a node is found to be conflict-free, it will be the final solution.

1.3 CCBS - Continuous Time CBS

Algorithm 2: Higher level of CCBS [Andreychuk et al., 2022]

```
Input: Graph $\mathcal{G} = (V, E)$, starting configuration S, goal
       configuration G
1  for $agent_i$ do
2      $\pi_i \leftarrow A^*(\mathcal{G}, S(i), G(i))$ ;
3  N.constraints = $\emptyset$;
4  N.solution = $(\pi_1, ..., \pi_k)$;
5  N.cost ← compute_cost(N.solution);
6  OPEN ← N;
7  while OPEN not empty do
8      N ← lowest_cost_node(OPEN);
9      if conflicts = $\emptyset$ then
10         return N.solution;
11     $(i, j, (a_i, t_i), (a_j, t_j))$ ← find_conflict(N.solution);
12     for agent $l$ in $\{i, j\}$ do
13         N$_l$ ← new_node();
14         $[t_l, t_l^u)$ ← unsafe_interval($l$);
15         N$_l$.constraints ← N.constraints $\cup \{l, a_l, [t_l, t_l^u)\}$;
16         $\pi_l'$ ← CSIPP($\mathcal{G}$, S($l$), G($l$), N$_l$.constraints);
17         N$_l$.solution[$l$] ← $\pi_l'$;
18         N$_l$.cost ← compute_cost(N);
19         Insert N$_l$ to OPEN;
```

The CCBS algorithm is analogous to CBS with the exception being the definition of a conflict and its resolution. In CCBS, the objective is also to minimise the makespan, defined as a maximum of costs of individual agents' paths. It is also a solution-complete algorithm, finding a solution if it exists, while unable to detect if a solution does not exist [Andreychuk et al., 2022].

In CBS, a conflict is typically represented as a tuple of an agent, a vertex, and a time step. As CCBS operates with continuous time, this is no longer viable. Instead of a vertex and a time step, CCBS takes into account an action of an agent l in a calculated $unsafe\ interval$ $[t_l, t_l^u)$. A CCBS conflict is defined using pairs of timed actions (a, t). Timed action means that an action a is executed from time t. A conflict of two agents i and j occurs if they execute their respective timed actions and collide. Therefore, a CCBS conflict is defined as a tuple $(i, j, (a_i, t_i), (a_j, t_j))$.

An unsafe interval of a timed action (a_i, t_i) is defined with respect to another timed action (a_j, t_j) as the maximal continuous time interval starting from t_i, in which if agent i would perform the action a_i, it would have a conflict with timed action (a_j, t_j).

As for computing unsafe intervals, exact computation may prove too computationally demanding. It is possible to search for a conflict several times with increasing times, and when no conflict is found, an endpoint of an unsafe interval is set. To further shorten that unsafe interval, a binary search can be used to find a more accurate endpoint.

Although initial path planning is done by A* algorithm [Hart et al., 1968] in CCBS, the CSIPP algorithm is used for re-planning given paths according to newly defined constraints.

For the low level of CBS, the authors proposed a modification of the SIPP algorithm, CSIPP - Constrained Safe Interval Path Planning. It is used to calculate safe time intervals based on constraints passed to it by CCBS. [Andreychuk et al., 2022]

1.3.1 SMT-CBS and COBRA algorithms

In [Surynek, 2020], a novel solution for the MAPF$_R$ problem was proposed. The SMT-CBS$_R$ provides a makespan-optimal solution to this problem considering continuous space and time with geometric agents. It combines CBS algorithm with satisfiability modulo theory using lazy construction of incomplete encodings. That way, not all constraints are introduced in the model, which makes it consider a subset of solutions. The produced solution can then be further refined if needed. This makes it possible for the solution to be found before complete specification of the model, thus saving computational time. Unlike the original CBS, SMT-CBS$_R$ does not branch after encountering a conflict, and instead refines the propositional model with a conflict elimination disjunctive constraint (a mutex). The results of this approach are comparable to those of CCBS.

The authors of [Čáp et al., 2015] proposed the COBRA algorithm, a complete decentralised algorithm for on-line multi-robot trajectory planning. Individual robots plan an optimal trajectory for themselves and synchronise information about trajectories with other robots via a distributed token-passing mechanism, where a token is used as a synchronised shared memory which holds current trajectories of all robots. The robots obtain relocation tasks from the user, and after obtaining a token which can be held by one robot at a time, it plans a trajectory to find a new best-response trajectory fulfilling the task. This trajectory needs to avoid collisions with the trajectories provided by the token. If this path is found, the token is updated and released to other robots, and the robot starts following its new trajectory. Once it reaches the goal, it can accept new relocation tasks. This approach was compared to an ORCA reactive technique and was shown to be up to 48% faster than this approach.

1.4 RRT and RRT* algorithms

A classic algorithm for path planning in a continuous metric space is the RRT algorithm (Rapidly-Exploring Random Trees) [LaValle, 1998]. A rapidly expanding random tree is constructed with iterative expansion by randomly sampling points that do not collide with an obstacle and connecting them to the nearest point in the tree. Expansions are biased towards unexplored parts of the search space. An RRT is probabilistically complete [LaValle, 1998], but the constructed paths are not guaranteed to be optimal. An improved version, RRT*, aimed to solve this problem by introducing the A* cost function into the original algorithm. Thus, the algorithm makes paths converge to optimal solutions [Karaman and Frazzoli, 2011].

In the original RRT, a new point is connected to its nearest point in the tree, while in RRT*, a point is added in a way that ensures its lowest possible distance from the start. To select a point to which to add, the k-nearest neighbours method is used to find several possible adepts. Then all possible ways to connect the new point via all of the neighbours are tried, and the path which proves to be the shortest is used, and the point is connected to the neighbour which lies on that path.

While RRT would end an iteration with connecting a new point, RRT* then goes on to validate the quality of paths of the new point's nearest neighbours. As was before done for the new point, an alternative path to all the neighbours leading through the new point is tried. If this path is shorter than the previous path, an edge that connected that neighbour to its original path is deleted, and a new edge is created in its place, connecting the neighbour and the newly added point. This method ensures that the algorithm's paths converge to optimal paths if an infinite number of nodes are added [Karaman and Frazzoli, 2011].

1.4.1 RRT* modifications

In [Čáp et al., 2013], the authors proposed a multi-agent variant of the RRT * algorithm, MA-RRT *. The agents in this model moved on a discrete motion graph in Euclidean space. As the environment was discrete, the graph version of RRT*, G-RRT* was used. The main difference between the two variants is the steering technique, which in G-RRT* is done using heuristic-guided greedy search. In MA-RRT*, greedy search generates a sequence of joint actions of all agents. After each action is generated, the set of actions is checked for possible conflicts. This approach was evaluated to be more efficient than A* in joint-state space when path finding is carried out in sparse, large environments, at the cost of slight solution quality.

[Li et al., 2014] introduced a RRT-A* variant for non-holonomic robots. Non-holonomic robots are constrained in a way which limits their movement. These kino-dynamic constraints pose a problem for motion planning, although the RRT algorithm is able to take them into account. However, it fails to optimize its planned paths. Therefore, the authors introduced the A* cost function into non-holonomic RRT and evaluated several metric functions. The improved algorithm effectively decreased path lengths, although it proved to not be efficient in environments containing local minima.

The authors of [Ayawli et al., 2019] presented ORRT-A*, an optimised version of the RRT-A* algorithm. Robots in this case move in a partially known environment, and therefore obtaining and processing of the map is necessary at the beginning. Then, an RRT road map is computed. Modified goal-biased RRT is used to ensure rapid generating towards the goal position. After obtaining an initial path, this path is further optimised using the A* heuristic function. After creating a path, it is smoothed via a cubic spline. Every data

Algorithm 3: RRT* [Karaman and Frazzoli, 2011]

```
   Input  : $V \leftarrow \{x_{init}\}; E \leftarrow \emptyset$; radius $r$
   Output: G=(V,E)
1  for $i = 1$ to $n$ do
2      $x_{rand} \leftarrow$SampleFree$_i$();
3      $x_{nearest} \leftarrow$Nearest(G=V,E), x$_{rand}$);
4      $x_{new} \leftarrow$Steer(x$_{nearest}, x_{new}$);
5      if $ObstacleFree(x_{nearest}, x_{new})$ then
6          $X_{near}$ = Near(G=(V, E), x$_{new}, r$);
7          $V \leftarrow V \cup \{x_{new}\}$;
8          $x_{min} \leftarrow x_{nearest}$ ;
9          $c_{min} \leftarrow$Cost($x_{nearest} + c$(Line($x_{nearest}, x_{new}$));
10         for $x_{near}$ in $X_{near}$ do
11             if $CollisionFree(x_{near}, x_{new}) \wedge Cost(x_{near} +$
                 $c(Line(x_{near}, x_{new})) < c_{min}$ then
12                 $x_{min} \leftarrow x_{near}$ ;
13                 $c_{min} \leftarrow$Cost($x_{near} + c$(Line($x_{near}, x_{new}$));
14             $E \leftarrow E \cup \{(x_{min}, x_{new})\}$ ;
15         for $x_{near}$ in $X_{near}$ do
16             if $CollisionFree(x_{new}, x_{near}) \wedge Cost(x_{new} +$
                 $c(Line(x_{new}, x_{near})) < c_{min}$ then
17                 $x_{parent} \leftarrow$Parent($x_{near}$);
18                 $E \leftarrow (E \setminus \{(x_{parent}, x_{near})\} \cup \{(x_{new}, x_{near})\}$;
19 return $G = (V, E)$;
```

point interval is represented using different cubic functions and in this way the interpolation technique smooths the path with the generation of more data points. During the course of a robot's motion in the environment, re-planning of its path is necessary to account for dynamic changes of the environment and newly discovered parts of the map. Experiments suggested that ORRT-A* enhances path quality in comparison to goal-biased RRT and RRT-A*.

The MA-RRT*FN algorithm was proposed in [Jiang and Wu, 2019], which aimed to increase the efficiency of the MA-RRT * algorithm by limiting the number of nodes stored in a tree to a fixed number. To avoid reaching this limit of nodes, all nodes are evaluated, and those which are unlikely to reach the goal are removed from the path. This solution does not exceed MA-RRT* concerning path quality, although the results are close, but it is much more efficient considering memory usage.

[Verbari, 2017] focuses on decentralized multi-agent path finding using RRT* algorithm. The author proposes Multi-RRT* algorithm, an approach aiming to find a Pareto optimal solution with using a priority order method. The problem is solved agent by agent by sorting them through a performance measure. It removes agents less involved in a collision from the problem, therefore solving each iteration in a smaller problem. Each agent computes its own trajectory with respect to solutions of other agents, which are considered as known, time-varying obstacles.

1.5 Path smoothing

In a real-life scenario, where an agent would be a vehicle, the vehicle would be sensitive for example to changes in acceleration, which would have to occur if it were to perform a sudden change of direction, such as a sharp turn. This could have an impact on the time that it spends on that path. Because of this, the differences in the different closely laid segments of the path should be limited [Lan and Di Cairano, 2015, Savla et al., 2005, Váňa and Faigl, 2015]. A possible solution is to introduce a smoothing mechanism, therefore making the path a smooth curve. The curvature of the path can then be constrained to prevent sudden turning changes. Because of these real-life implications, it is advantageous to work not only with continuous time, but with continuous environment, too, as it can introduce movement on smooth curves and therefore take various kinematic or dynamic constraints into account.

A path generated by RRT* can contain sharp turns when going from one path segment to another. To improve the quality of a path, many smoothing mechanisms were proposed, starting with [Dubins, 1957]. This work uses smoothing via B-spline curves, so as to be able to represent the complex nature a path in an environment with many obstacles can have. A B-spline curve is fitted onto a path with concrete path points being curve control points, and additional points are then interpolated laying on the smooth curve.

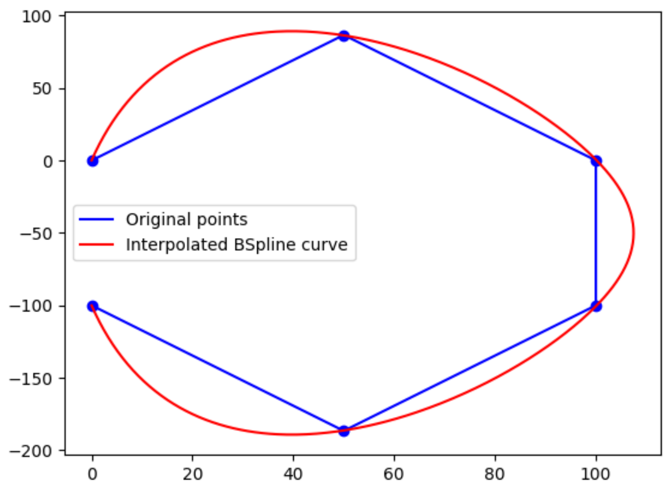


Figure 1.1: An example of how a B-spline curve is created on a path consisting of 6 points.

1.5.1 B-spline curves

A B-spline curve is a piecewise polynomial curve. A spline is a piecewise polynomial of degree n with segments C^{n-1}. It can be thought of as a method to define a sequence of degree n Bézier curves that automatically join with C^{n-1}, regardless of the placement of the control point. It can be used in problems too complex to be modelled with a Bézier curve. [Sederberg, 2012]

Definition 1.5.1 (B-spline basis function)**.** Let $U = \{u_0, ..., u_m\}$ be a non-decreasing sequence of real numbers. The u_i are called knots, and U is the knot vector. The i-th B-spline basis function of p degree, denoted as $N_{i,p}(u)$ is defined as

$$\begin{aligned} N_{i,0}(u) &= \begin{cases} 1 & u_i \leq u < u_{i+1} \\ 0 & otherwise \end{cases} \\ N_{i,p}(u) &= \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \end{aligned} \tag{1.1}$$

[Piegl and Tiller, 1996]

A B-spline curve is then defined as a linear combination of control points and B-spline basis functions.

Definition 1.5.2 (B-spline curve)**.** A p-th degree B-spline curve is defined as:

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i \tag{1.2}$$

where $a \leq u \leq b$, P_i are control points, and $\{N_{i,p}(u)\}$ are p-th degree B-spline basis functions defined on non-periodic knot vector $U = \{a, ..., a, u_{p+1}, ...,$ $u_{m-p-1}, b, ..., b\}$. [Piegl and Tiller, 1996]

1.5.2 Existing path planning algorithms using smoothing

[He et al., 2016] focused on MAPF with continuous time. Agents are controlled globally and use general linear dynamics. The authors focus on navigating agents through complicated environment such as crowded areas or narrow spaces. These challenging parts are called bridges, and they represent collision-free regions of the environment with certain geometric navigation characteristics. Trajectory of each agent is computed with kinodynamic RRT*, and it moves an agent along an interpolated path in the bridge. It is possible for multiple agents to be present at one bridge at the same time, as a scheduling system is present to ensure efficient and collision-free motion.

In [Cao et al., 2023], a novel approach to path smoothing was proposed, called the point adjustment algorithm and smoothing. Agents in the modelled environment moved on a rasterised 2D space and were represented as a mass point in space. After the initial path is obtained via a path planning algorithm, a path point adjustment is run. This is done to increase the angles between the path segments and also to reduce the length of the original path. This is done with the line-of-sight algorithm, which removes unnecessary turns in a path. Then, a rotate line algorithm is run which enlarges the angles between path segments by adjusting the positions of path points. The last step of point adjustment is the parallel line algorithm, which ensures that all angles on a path follow a minimum angle requirement, 90 degrees. The second part of this algorithm, turn-smoothing, smooths a processed path via B-spline curves. B-spline curves are inserted in turns, and the control points of the curve are adjusted for the path to meet certain robotic requirements, path continuity, path safety, and maximum path curvature.

The authors of [Eshtehardian and Khodaygan, 2022] worked with non-holonomic vehicles in a continuous environment and proposed a model for smooth movement with combination of RRT* for path planning and B-spline curves for additional path smoothing. The initial path planning used RRT*, whose result was then optimised to account for non-holonomic constraints restricting the turn angle of assumed vehicles. Therefore, some nodes of this initial path were removed, which made for a smoother and shorter path. Then, B-spline interpolation was used to smooth the path by generating a curve, thus fulfilling non-holonomic constraints.

The authors of [Neto et al., 2010] proposed an approach to path planning for non-holonomic vehicles using seventh-order Bézier curves. This approach allows for the use of simple kinematic and dynamic constraints and ensures path smoothness. The main constraint considered in this model is the minimal curvature that a vehicle can execute. The path is generated by a variant of the RRT algorithm. Curves are established between the states generated by RRT. These curves are compositions of seventh-order Bézier functions that satisfy all the constraints considered. This approach reduces the number of vertices on a path, especially in an environment with a low number of obstacles, where many vertices can become obsolete.

In [Garrido Carpio, 2018], the author proposed a local path planning model that combines pre-planning and real-time planning, which generates continuous paths in order to minimise curvature and abrupt changes. The first step of this model is to preplan optimal Bézier curves considering physical and kinematic constraints. Then, a real-time algorithm is applied, resulting in a continuous path which is created by joining the pre-planned curves. It can be used in both static and dynamic environments, as the dynamic path planning algorithm aims to generate smooth avoidance paths efficiently.

[Lai et al., 2023] proposed improved A* algorithm, which aims to shorten the computational time and reduce redundant nodes, adding a dynamic weighting factor to the heuristic function of A*. This allows for dynamic focus change during the course of the algorithm. Then, a fusion algorithm was proposed which combined this improved A* and segmented second-order Bézier curves, resulting in smooth paths.

Chapter 2

Proposed model

2.1 CBS in continuous environment

This works builds on the Continuous Conflict-Based Search algorithm proposed by [Andreychuk et al., 2022]. Here, a modification called Continuous Environment Conflict-Based Search (CE-CBS) is proposed. Its aim is to minimise the necessary discretization of the environments. It operates in a continuous environment with continuous time. Each agent in this multi-agent simulation performs path finding via the RRT* algorithm proposed by [Karaman and Frazzoli, 2011], which is modified to both solve constraints provided by CE-CBS and provide a path which is then smoothed using B-spline curves. The goal is to find a conflict-free solution while minimising the criterion *sum of costs*, which is the total sum of the paths of all agents.

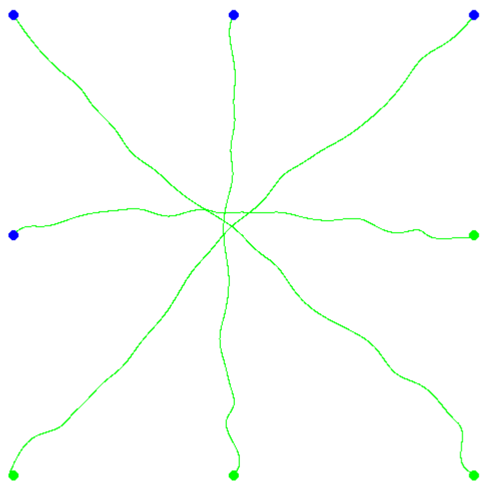


Figure 2.1: A visualization of a non-conflicting set of smooth paths (in green) for a set of agents going from their start positions (blue dots) to their goal positions (green dots).

Agents in this model are defined as circular bodies with fixed radius and speed, both of which are input parameters. They are modelled as goal-based agents [Russell and Norvig, 2020] moving in a known environment. This model can be seen in figure 2.2. These agents aim to plan a set of paths from start positions to goal positions while avoiding both static obstacles, which are represented here as rectangles of varying sizes, and dynamic obstacles, which are constraints set by CE-CBS to avoid collisions between agents.

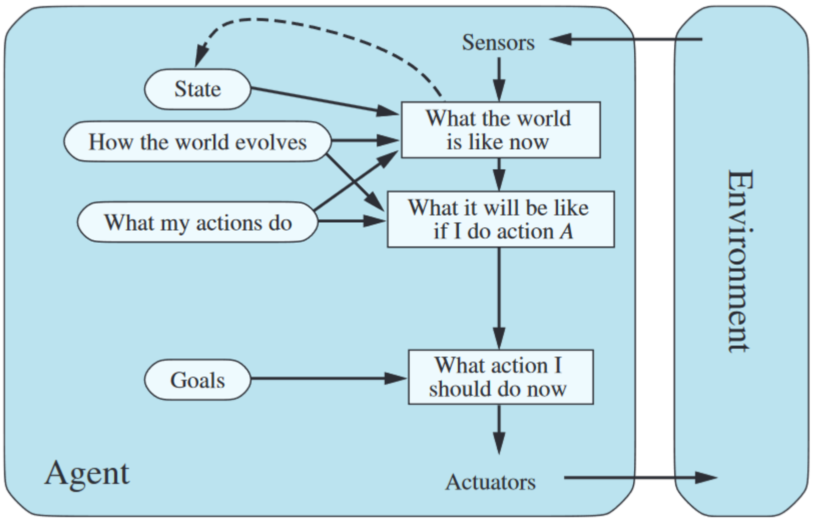

Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

Figure 2.2: A goal-based agent as defined in [Russell and Norvig, 2020]

In the base design of this work, agents are set to find smooth paths in a way that they can move continuously without ever stopping and waiting before reaching their goal position.

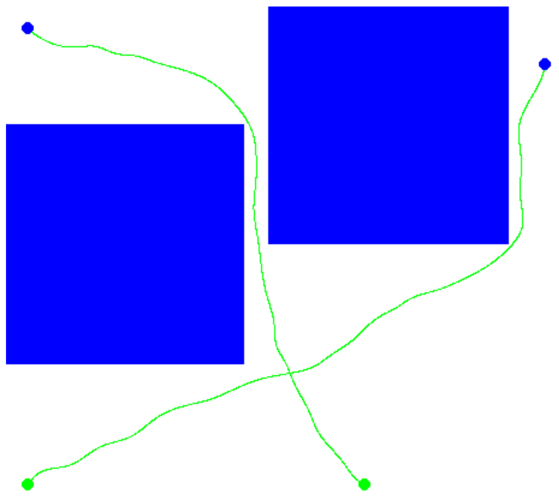


Figure 2.3: A case where two agents successfully plan non-conflicting smooth paths from their start positions (blue) to their goal positions (green). Their respective paths are highlighted green.

As with CCBS, the cost of a solution is the sum of the costs of each agent's individual path. [Andreychuk et al., 2021] The length ρ of a path is measured as the sum of distances between neighbouring points on a path. The metric used is the Euclidean distance:

$$\rho_i = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{2.1}$$

The goal of the algorithm is to minimise this sum of costs while keeping the plan conflict-free.

2.1.1 Redefining conflicts

A conflict in this work is defined as a 5-tuple $C = (a_i, t_{a_i}, a_{i+1}, t_{a_{i+1}}, p_i, p_{i+1})$, where a_i, a_{i+1} are two conflicting agents, $t_{a_i}, t_{a_{i+1}}$ are the respective times of the agent to reach the conflict position and p_i, p_{i+1} are the conflict coordinates for each agent.

Whether two agents are in a conflict is determined in two steps. For each pair of agents, it is checked if their planned trajectories intersect. If so, a time is computed that represents the moment when the two agents reach the intersection. This is done with the basic formula of $t = \frac{s}{v}$, where t is the resulting time, s is the length of the agent's path from the start to the intersection point, measured as a sum of Euclidean distances for each path segment, and v is a constant speed of an agent.

It is also necessary to account for agent bodies, for which checking if paths intersect does not suffice. It is therefore also checked if the distance between every pair of path segments from different paths is closer to each other than the agent radius. If so, the points at which those segments are closest are tested to see at which time agents arrive at these positions. If these times fall into an unsafe interval of another agent, a conflict is present.

An unsafe interval is created from these time variables, which is necessary to take the agent mass into account. If the time when agent a_1 reaches the intersection point falls into the unsafe interval of agent a_2 or vice versa, a conflict is present.

Algorithm 4: Get first conflict

```
   Input  : paths $P$, agents $A$
   Output: conflict $c$
1  $C \leftarrow \emptyset$;
2  for $a_i \in A$ do
3      for $a_{i+1} \in A$ do
4          $crossing, positions \leftarrow$ CrossingCloseTrajectories($P_{a_i}, P_{a_{i+1}}$);
5          if $crossing$ then
6              for $pos_i, pos_{i+1} \in positions$ do
7                  $t_{a_i} \leftarrow$ TimeToReach($P_{a_i}, pos_i$);
8                  $t_{a_{i+1}} \leftarrow$ TimeToReach($P_{a_{i+1}}, pos_{i+1}$);
9                  if $TimesTooClose(t_{a_i}, t_{a_{i+1}})$ then
10                     $C \leftarrow C \cup$ Conflict($a_i, t_{a_i}, a_{i+1}, t_{a_{i+1}}, pos_i, pos_{i+1}$);
11 return SelectFirst(C);
```

The conflict which is returned by the *GetFirstConflict* function as seen in algorithm 4 is that in which one of the agents' time is the lowest from all the times of all conflicts.

2.2 RRT*

The lower level of CE-CBS is performed by RRT* enhanced by several steps from the original algorithm proposed by [Karaman and Frazzoli, 2011]. The lower level, as seen in the algorithm 5, operates in two basic steps: path planning and interpolation. Each part has to be checked for validity to ensure the resulting path being as conflict-free and obstacle-free as possible, although it returns a path breaking these constraints if no other was found.

Algorithm 5: Lower level of CE-CBS

```
  Input : agent $a$, obstacles $O$, constraints $C$, maximum nodes
          value $m$
  Output: path $p$
1 $valid \leftarrow$ False;
2 $p \leftarrow \emptyset$;
3 $n = m$;
4 while not valid do
5     $p, valid \leftarrow$RRTstar$(a, O, C, n)$;
6     if $valid$ or $n \leq m/5$ then
7         $p, valid \leftarrow$Interpolate$(p, O, C)$;
8     $n = n - m/10$;
9 return $p$;
```

The first step is the RRT* path planning, which in itself validates the path as described below. If the path found is not valid, meaning not only that it is conflict-free and obstacle-free, but that all angles between neighbouring path segments are larger than 90 degrees, RRT* is re-planned with a lower value of maximum nodes to sample. If a valid solution is found, interpolation with a B-spline curve can take place. The path resulting from interpolation is then validated in the same manner as the path resulting from RRT*, and interpolation can also be done multiple times with different values of the smoothing parameter s to try and find a sufficient path without the need to re-planning the entire path by RRT*.

Figure 2.4 shows the result of RRT* before smoothing. The start of the path is noted by a red dot and the goal location by a green dot. All the sampled coordinates with lines connecting them to the tree are drawn in red while the final path, found as the shortest from all available paths, is drawn green. Although RRT* converges to optimal solution [Karaman and Frazzoli, 2011], that solution may not be found in cases where not enough nodes are sampled. This work counts with sub-optimality of solutions provided by RRT*, as the computational time has to be taken into account.

Experiment-based modifications Based on result of experiment 3.4.8, which studies various values of parameter α_{min}, which denotes the minimal

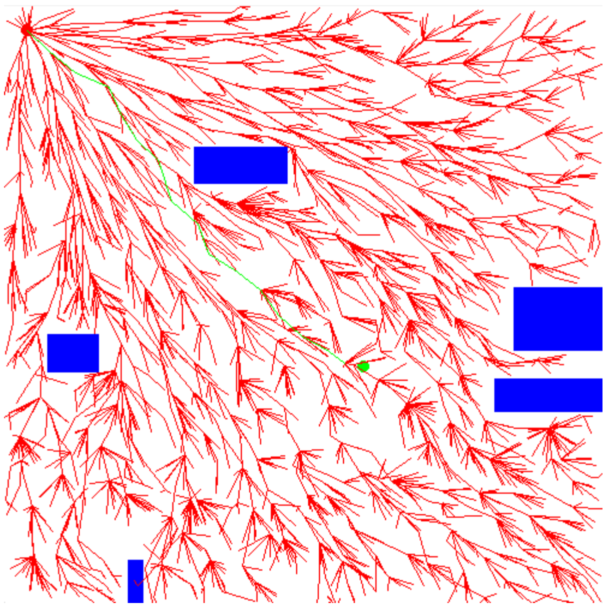


Figure 2.4: An example of RRT* algorithm. Red lines represent all edges in the graph and a green line represents the final path.

angle all neighbouring path segments must hold, algorithm 5 was modified to ensure valid results are returned for instances in a difficult setting. The experiment showed it is necessary to be able to dynamically set the lower bound of maximal number of nodes expanded by RRT* η_{max}, so that the entire environment can be searched even if this value is lowered. Therefore parameter η_{min} is introduced, representing the denominator used for counting the lower bound. Another newly introduced parameter is parameter σ, which controls the rate by which η_{max} is lowered. To ensure that RRT* does not return a solution which is either incomplete or breaking an angle constraint, iterating continues after reaching the lower bound, but it does not lower η_{max} further. This modification only impacts cases where a valid solution wouldn't be found otherwise, and is used in experiments 3.4.9 and below. The modified algorithm is showed in algorithm 6.

2.2.1 CE-CBS and kinematic constraint satisfaction

The RRT* algorithm takes a set of constraints on input. A constraint is defined as a 4-tuple $c = (a, p, unsafe_{from}, unsafe_{to})$, where a specifies an agent to which that constraint applies, p is a particular coordinate of a conflict and $(unsafe_{from}, unsafe_{to})$ denotes the unsafe time interval. If an agent crossed p in a time within the unsafe interval, it would mean breaking that constraint.

While path planning, RRT* has to take two types of obstacles into account – both static predefined rectangular obstacles which remain obstacles throughout the whole course of the planning, and constraints provided by CE-CBS, which serve as dynamic obstacles and only present a threat in a certain time frame specified by the unsafe interval.

Algorithm 6: Modified Lower level of CE-CBS

```
Input  : agent $a$, obstacles $O$, constraints $C$, maximum nodes
         value $m$
Output: path $p$
1  $valid \leftarrow$ False;
2  $p \leftarrow \emptyset$;
3  $n = \eta_{max}$;
4  while not valid do
5      $p, valid \leftarrow$RRTstar$(a, O, C, n)$;
6      if valid then
7          $p, valid \leftarrow$Interpolate$(p, O, C)$;
8      if $n > \frac{\eta_{max}}{\eta_{min}}$ then
9          $n = n - \frac{\eta_{max}}{\sigma}$
10 return $p$;
```

If a path segment is to be added to the RRT* tree, it first needs to be checked for validity, so that the line segment formed by adding a new point does not intersect an obstacle or break a constraint.

Algorithm 7: Validate

```
Input  : path $p$, obstacles $O$, constraints $C$
Output: True or False
1  $segment \leftarrow (p_0)$;
2  for $i$ in $length(p) - 1$ do
3      $segment \leftarrow segment + p_{i+1}$;
4      $line \leftarrow p_i, p_{i+1}$;
5      if not $ObstacleFree(line, O)$ then
6          return False;
7      $cost \leftarrow$PathCost$(segment)$;
8      if not $ConstraintFree(line, cost, C)$ then
9          return False;
10 return True;
```

The algorithm 7 shows the function *validate*. This function checks if any line segment defined by two neighbouring points intersects an obstacle or a constraint. Intersecting a constraint means that the agent would arrive at a given position in a time specified by the unsafe interval of the constraint.

2.2.1.1 Minimum angle constraint

A sufficient path must satisfy the given kinematic constraints. In this work, angles between path segments (pre-smoothing) are constrained to be bigger than 90 degrees to ensure that a simulated robot would be able to preform the turn. This constraint number was preliminarily chosen so as to eliminate acute angles from paths and can be changed in further experiments.

Using the cosine formula for the dot product, the angle θ between two line segments represented by two vectors v, w can be written as:

$$\theta = cos^{-1}(\frac{v \cdot w}{||v|| \cdot ||w||}) \tag{2.2}$$

When RRT* planning starts, it is given a constraint for the maximum number of nodes it is able to sample. This number is provided in the input. RRT* is the run with this number. Preliminary experiments have shown that in some edge cases where an obstacle is narrow and a sharp turn is path length optimal, a path with an acute angle will be generated as seen in figure 2.5.

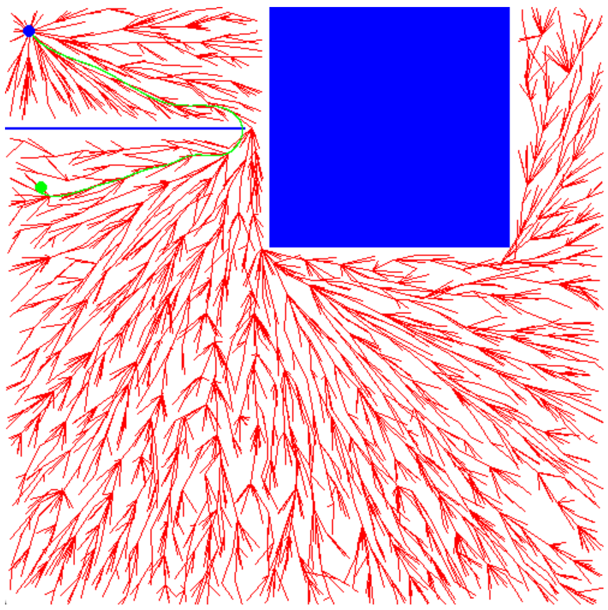


Figure 2.5: A path with an acute angle leading to collision with an obstacle after smoothing.

A path segment can be modified in a number of ways to correct this, as described below, but this approach may sometimes not suffice. In that case, a new iteration of RRT* is run with a lower number of maximum node number. This iteration may opt for a less optional path, but this setback can be beneficial in keeping all angles larger than the limit. If one re-planning does not suffice, more iterations follow, each with a lower maximum node parameter value, until an obstacle and conflict-free path has been found or a minimal value of this parameter is hit, if ending the search this way is allowed. Figure 2.6 shows a path successfully navigating in a problematic environment.

When a path is constructed by RRT*, it is validated before being passed on to be smoothed. This validation checks all angles between pairs of neighbouring path segments and tries to correct them if they are too small. This method is shown in Algorithm 8. If an acute angle (in the base design, a different bound can be used as a minimal angle, as shown in lines 3 and 6) is found, the algorithm first tries to eliminate the vertex which is shared by the two path segments and connect the other two vertices instead - checked by function $Eliminate(path, index, obstacles, constraints)$ on line 4. If a vertex can

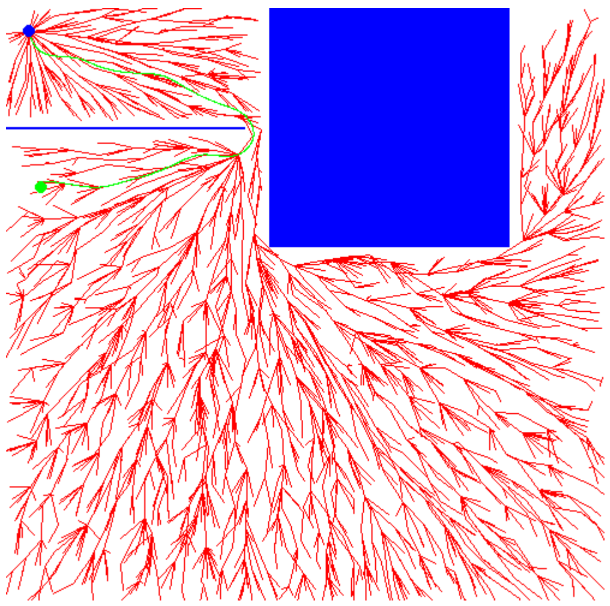


Figure 2.6: A smooth path avoiding a narrow obstacle successfully.

be eliminated, it is not added to the new path q. This will not only shorten the original path but will account for increasing the previously acute angle to 180 degrees. This is only possible if there are no obstacles or constraints preventing connecting the other two path segments.

If a shared vertex cannot be eliminated, the algorithm then tries to move it closer to the other two vertices to increase the angle size. This is done by the function $MoveCloser(path, index, obstacles, constraints)$ on line 5. The point moves along a perpendicular to line segment formed by the other two vertices. The final position must be as close to the intersection point of these lines as possible without colliding with an obstacle or violating any constraints.

If neither of these approaches enlarges the angle enough, RRT* runs again with a smaller value of the maximum node value to find a possibly less optimal path that may satisfy the constraint.

2.2.2 Path smoothing

As with the path planning itself, smoothing takes several steps. The interpolation itself is performed first. Given path points resulting from RRT*, a B-spline curve representation is found using the *splrep* function of the *scipy* library [SciPy, 2023b]. Knots and coefficients returned by this function then serve as input for the *splev* function from the *scipy* library [SciPy, 2023a], which returns $length(path) * max\_distance$ points along this B-spline curve, where $max\_distance$ is the highest distance in which a point can be sampled from its nearest neighbour; this way, a new point is interpolated for at least each unit of path length. Although a smooth path is found at this stage, it still needs to be validated using the validate function as shown in the algorithm 7 to check if this new representation satisfies all constraints and avoids all obstacles. If it does not succeed, the interpolation is then tried again with a larger value

Algorithm 8: Adjust Angles

```
Input  : path $p$, obstacles $O$, constraints $C$, angle $a$
Output: True or False, path $q$
1  $q \leftarrow (p_0)$;
2  for $i$ in $length(p) - 2$ do
3      if $Angle((p_{i+1}, q_{last}), (p_{i+1}, p_{i+2})) < a$ then
4          if not $Eliminate(p, i+1, O, C)$ then
5              $moved\_pos \leftarrow MoveCloser(p, i+1, O, C)$;
6              if $Angle((moved\_pos, q_{last}), (moved\_pos, p_{i+2})) < a$ then
7                  return $(False, p)$;
8              $q \leftarrow q + moved\_pos$;
9      else
10         $q \leftarrow q + p_{i+1}$;
11 $q \leftarrow q + p_{last}$;
12 return $(True, q)$;
```

of the smoothing parameter s, which is initialised as $s = 0$. This is repeated if necessary for $s \leq 5$.

If no path is found, even then, the interpolation ends with a fail. If RRT* did not previously plan its path with a borderline low value of maximum node number η_{max}, it can be replanned with a smaller value and interpolated again, as shown in the algorithm 5. After modifying the algorithm as discussed in section 2.2, the algorithm continues to search for a valid solution even after decreasing η_{max} to the minimum allowed number η_{min}. The search continues with $\eta_{max} = \eta_{min}$ until a valid solution is found.

Figure 2.7 shows how a smooth path resulting from RRT* and subsequent interpolation looks. RRT* with smoothing performed well in this case concerning a narrow passage where collisions with corners of objects would be possible.

2.3 Implementational details

The implementation of this model consists of six main classes. The *CECBS* class operates the higher level of the CE-CBS algorithm and stores nodes of the conflict tree. Each node is an object of class *CECBSNode*, which directs the low-level search of the agents' paths and stores their solutions. The low-level search takes place mainly in the *RRTstar* class, which contains all the methods necessary to calculate the path of an individual agent. The main method in this class is the *rrt_star*() method, whose results is a non-conflicting path, which is yet to be smoothed. The nodes of the RRT tree are represented by the *RRTNode* class. All smoothing takes place in the functions located in the module *support_functions.py*. The environment itself is stored in the *RRTMap* class, which also orchestrates visualisation of the map and found paths. The parameters passed to the algorithm from a configuration file, such as the maximal amount of RRT* node η_{max} or agent radius r are stored in the *Params* class. The UML class diagram is shown in Figure 2.8.

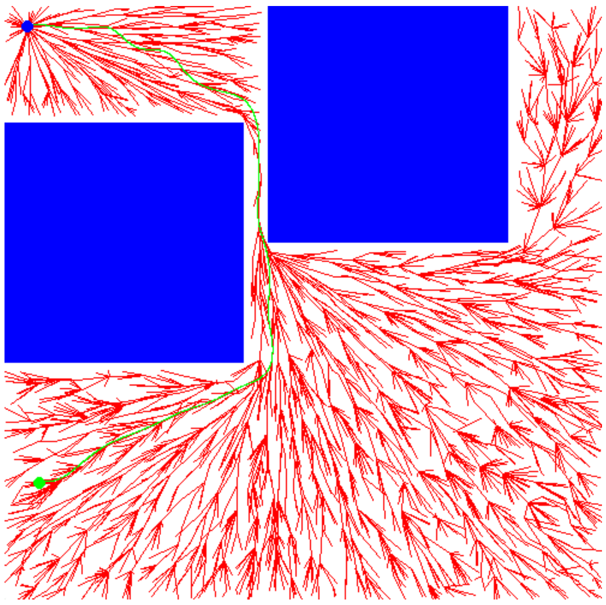


Figure 2.7: An example of RRT* algorithm with the green path already being smoothed. Red lines represent all lines in the RRT* tree.

This model has been programmed in the Python programming language using Numpy, SciPy, Shapely, and PyGame libraries. The SciPy library is used for the B-spline computation and interpolation, while Shapely and Numpy libraries are used for the computation of geometric elements, for example, checking the distance between path segments and obstacles or providing supporting calculations during computations of angles between path segments. The PyGame library serves for the visualisation of the environment, consisting of obstacles, agent start and goal coordinates, and the paths between said coordinates.

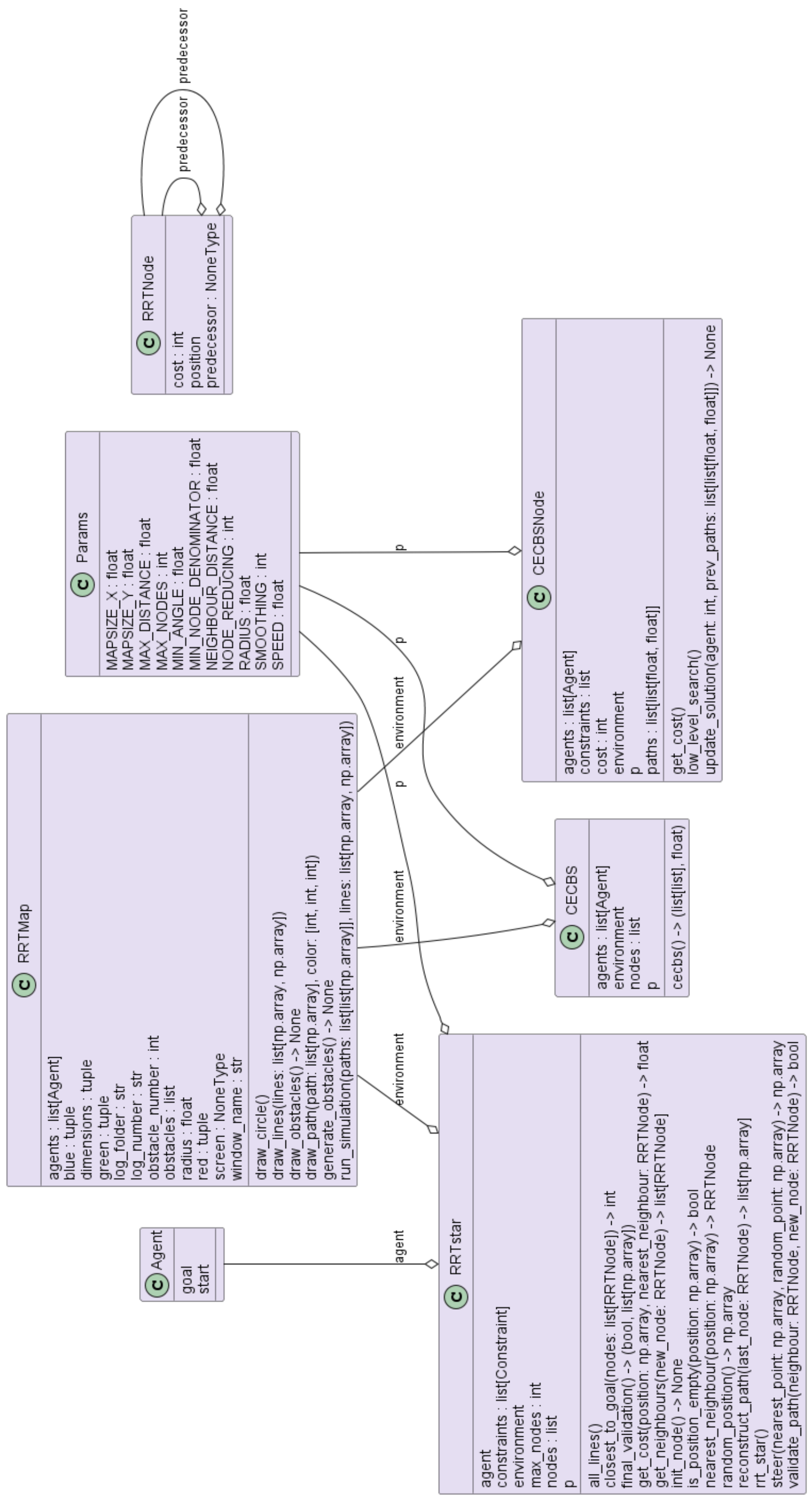

RRTNode
cost : int
position
predecessor : NoneType
predecessor
predecessor
Params
MAPSIZE_X : float
MAPSIZE_Y : float
MAX_DISTANCE : float
MAX_NODES : int
MIN_ANGLE : float
MIN_NODE_DENOMINATOR : float
NEIGHBOUR_DISTANCE : float
NODE_REDUCING : int
RADIUS : float
SMOOTHING : int
SPEED : float
CECBSNode
agents : list[Agent]
constraints : list
cost : int
environment
p
paths : list[list[float, float]]
get_cost()
low_level_search()
update_solution(agent: int, prev_paths: list[list[float, float]]) -> None
p
p
environment
p
CECBS
agents : list[Agent]
environment
nodes : list
p
cecbs() -> (list[list], float)
environment
RRTMap
agents : list[Agent]
blue : tuple
dimensions : tuple
green : tuple
log_folder : str
log_number : str
obstacle_number : int
obstacles : list
radius : float
red : tuple
screen : NoneType
window_name : str
draw_circle()
draw_lines(lines: list[np.array, np.array])
draw_obstacles() -> None
draw_path(path: list[np.array], color: [int, int, int])
generate_obstacles() -> None
run_simulation(paths: list[list[np.array]], lines: list[np.array, np.array])
environment
Agent
goal
start
agent
RRTstar
agent
constraints : list[Constraint]
environment
max_nodes : int
nodes : list
p
all_lines()
closest_to_goal(nodes: list[RRTNode]) -> int
final_validation() -> (bool, list[np.array])
get_cost(position: np.array, nearest_neighbour: RRTNode) -> float
get_neighbours(new_node: RRTNode) -> list[RRTNode]
init_node() -> None
is_position_empty(position: np.array) -> bool
nearest_neighbour(position: np.array) -> RRTNode
random_position() -> np.array
reconstruct_path(last_node: RRTNode) -> list[np.array]
rrt_star()
steer(nearest_point: np.array, random_point: np.array) -> np.array
validate_path(neighbour: RRTNode, new_node: RRTNode) -> bool

Figure 2.8: UML class diagram.

Chapter 3

Experimental evaluation

This part focuses on the model's behaviour in several types of environment. It evaluates both performance depending on the number of agents and parameters, such as the maximal number of nodes RRT* can expand. This is evaluated using both the standard sum of costs and computational time. This is done to find out how path re-planning affects the computation. The individual costs of agent paths are also discussed in some experiments concerning an increasing number of agents in one plan, whose individual optimal path lengths would be equal.

3.1 Parameters

Maximal number of nodes in RRT* η_{max}: The value of this parameter sets an upper boundary to expansion of the RRT* tree. Increasing this value may lead to increased computational time, but also to a path closer to the optimal path. Preliminary experiments have shown that, in some cases, more optimal paths can create sharp turns unsolvable by node adjustment, and therefore paths have to be re-planned, further leading to increased runtime. If more iterations of RRT* with decreasing η_{max} are run, the speed at which it decreases is controlled by the reducing rate σ, which is in the base design set to $\sigma = 10$. If a current value of the maximum number of RRT* nodes in a given iteration is denoted as n, the formula by which it decreases is set as $n = n - \frac{\eta_{max}}{\sigma}$. The value of n is initialised as η_{max}.

Minimal number of nodes in RRT* η_{min}: This parameter sets the lower boundary to the expansion of the RRT* tree. It is used in a case where multiple iterations of re-planning have taken place. Setting this value too low can prevent the algorithm from finding a path, or the algorithm can find a path that much less optimal than the shortest path. Lowering the number of expanded nodes may lead to decreased computational time. The value of this parameter is tied to η_{max}, as its value is set as $\eta_{min} = \frac{\eta_{max}}{5}$.

Minimal angle in a path α_{min}: Denotes an angle boundary. If an angle between two adjoining path segments is below this boundary, this angle has to be adjusted if possible. If this is not possible, re-planning has to occur. In the base design of this model, this value is set as $\alpha_{min} = 90°$, prohibiting acute angles on paths and therefore preventing agents from taking sharp turns.

Number of agents n: The number of agents in an environment. All of these agents participate in CE-CBS and are goal-based agents with knowledge of the environment. Each one of them starts in a unique location and has a unique goal location. The utility function used to evaluate their paths is the sum of costs, or in some cases, average of the costs if this number varies between individual iterations of this algorithm.

Smoothing parameter s: The parameter s is used in smoothing a path resulting from RRT*. It is used to control the trade-off between the smoothness of the path and the closeness to the original path. Higher s means more smoothing, thus possibly straying further from the original path [SciPy, 2023b].

Agent radius r: This parameter determines the size of agent body, which is circular. Increasing the size of agent body may make it more difficult to plan a set of paths where no agents overlap at any time, as it is necessary to check not only if their trajectories cross, but the possible body overlap in cases, where different agents' path segments get close to each other.

3.2 Maps

Open environment: An environment without any static obstacles is experimented with to determine the quality of the path in cases where a straight line with length denoted by a Euclidean distance from a starting position to a goal position would be the optimal solution (not concerning other agents). These experiments are done to determine how path quality in this model differs from the optimal solution and how it evolves, when it becomes necessary to modify some of the paths to account for solving conflicting paths, which inevitably lead to modifying at least one of the original paths.

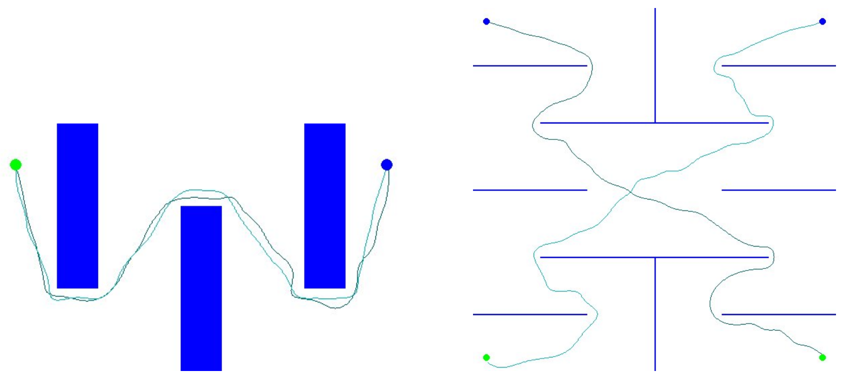

(a) A map with several obstacles (b) A maze map with narrow obstacles

Figure 3.1: Examples of two maps with multiple agents crossing through them.

Several large obstacles: In an environment pictured in figure 3.1a, two agents are moving along a similar path, but in opposite direction. The starting position of one agent is the other agent's goal position. This means that approximately in the centre of the map, agents meet and have to avoid each other without their bodies overlapping. This map is therefore used for testing among other parameters agent behavior based on agent radius r.

Maze: This type of environment can be used to determine ideal values for some of the parameters such as minimal angle α_{min}, which, as discussed in Section 2.2.1.1, can encounter problems in plans where a sharp turn would make a path shorter. In figure 3.1b, a maze map is pictured. Many narrow obstacles are present, and agents are forced to make several turns in opposite directions, creating a possibility of breaking angle constraints.

Narrow passages: Maps with one narrow passage represent a situation where there are limited possibilities of movement, thus making avoiding other agents more difficult. Agents will be forced to plan paths avoiding each other without overlapping at any time, which can prove challenging in a narrow space.

3.3 Measures of solution quality

Sum of costs (SOC): In this work, the objective is to minimize the *sum of costs* of all agents' paths, while keeping the set of paths collision-free. This measure is used in a majority of experiments where the number of agents does not differ between experiments. For experiments where number of agents n vary, an average of SOC per agent is made and compared. Another method which can be used to evaluate the plan quality is the makespan, which is the maximum of all agents' individual path costs.

Scatter ζ: To evaluate consistency of each plan, it is studied how different solutions of one instance vary. Lesser ζ means the algorithm is able to reproduce a result close to an average result more often. With higher ζ, results run on the same one instance differ more than if there would be lower scatter. For example, in figure 3.6 belonging to experiment 3.4.3, high ζ can be observed with lower values of η_{max}, particularly with $\eta_{max} = 1000$, decreasing significantly afterwards.

Number of CE-CBS iterations and execution time: From a technical standpoint, the total execution time of the algorithm is an important measure, as it can be telling about the computational load of the model. Preliminary experiments have shown that the computational time depends largely on the number of CE-CBS iterations run, as the part that takes the longest to compute is the method checking the validity of a CE-CBS solution. Due to the hardware limitations while testing, execution times can vary from a couple of minutes to several hours, depending on the difficulty of problem that is being solved. As some experiments are run on multiple machines, it can be noted that the execution time is hardware dependent. For this reason, mainly the number of CE-CBS iterations is discussed in this chapter. The model takes less execution time when running on the CloudFIT platform [FIT ČVUT, 2024] than when running on the author's machine with AMD Ryzen 5 5600H CPU and 16GB RAM.

The number of CE-CBS iterations can also be interesting from the point of view concerning path quality, as in cases with many conflicts being solved, the space becomes more constrained, allowing for example RRT* to search the non-constrained parts more.

3.4 Experiments

3.4.1 Empty space with increasing number of agents - grid pattern

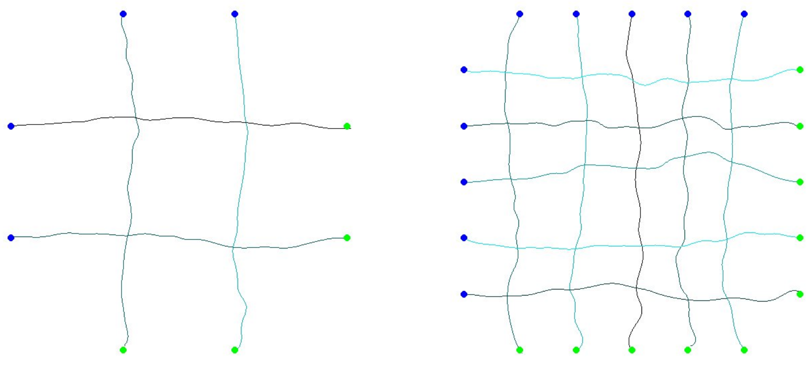
The first experiment shows how an increasing number of agents n affects the individual path lengths of the agents in an environment without static obstacles. Each of the agent's paths has the same optimal length denoted by Euclidean distance from a start position to a goal position, not considering other agents.

Environment dimensions	$(540, 540)$
Number of agents n	$\{2, 4, 6, 8, 10\}$
Max nodes η_{max}	4000
Min nodes η_{min}	$\eta_{max}/5$
Reducing rate σ	10
Min angle α_{min}	90
Smoothing s	0
Radius r	5

Table 3.1: Parameters of experiment 3.4.1 concerning parameter n

Parameter setting The tested values of n were $2, 4, 6, 8$ and 10. For each of these values, 20 iterations were run. These are shown in the box plot below.

Hypothesis As the number of agents grows, it should become more complicated for them to avoid collisions, thus needing more iterations of CE-CBS to be run. The collisions created by them may prevent agents from taking the shortest path possible, and thus with a higher number of agents, the average path length should rise.



(a) A case with $n = 4$ (b) A case with $n = 10$

Figure 3.2: Output examples for two different values of n. Agent starting positions are marked as blue points, goal positions as green points and paths as green lines.

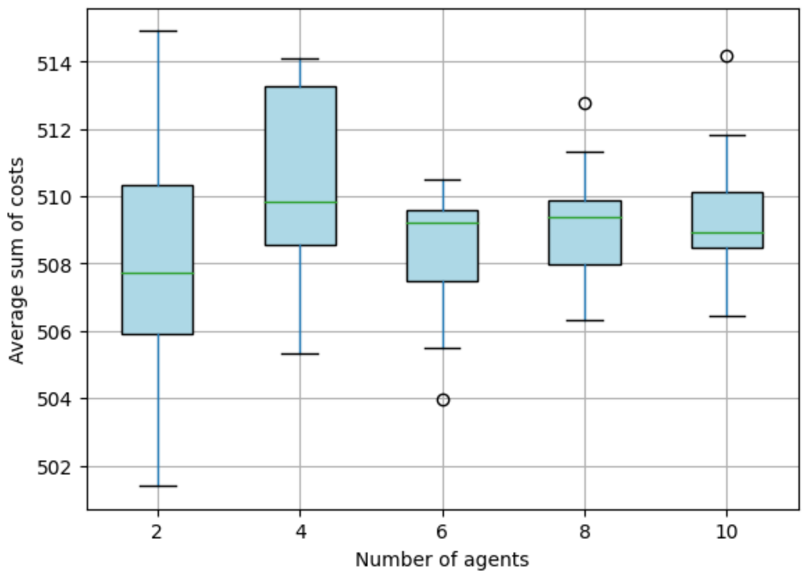


Figure 3.3: Boxplot graph depicting results of experiment 3.4.1. Each box is for a different number of agents, with y axis representing average path costs for one agent.

Results Figure 3.3 shows a boxplot graph resulting from this experiment. The graph shows that in cases with fewer agents, such as $n = 2$, lower average path costs were achievable, although the values were more scattered compared to higher values such as $n = 8$ or $n = 10$.

Interpretation Although lower average path lengths were achieved, the lower the value of n, the worst-case path length first decreased with increasing n, then began to rise again with $n > 6$. It is possible that changing the values of some parameters, such as the maximum number of RRT* nodes η_{max} could affect the scatter that occurred during experiments in the lower n. Further experiments below focus on examining this possibility.

3.4.2 Empty space with increasing number of agents - star pattern

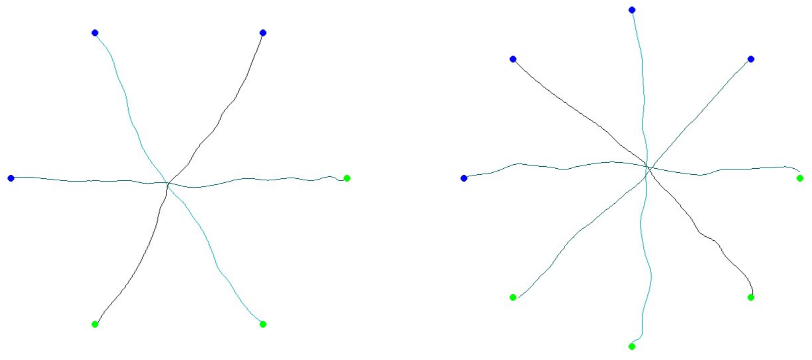
Smaller numbers of agents were used in this experiment compared to experiment 3.4.1, as the goal of this experiment was to test a scenario in which all agents should collide at the same time in the same position, if they planned an optimal path for themselves. Preliminary experiments have shown that for more than 4 agents, this is a computationally expensive task, as large numbers of conflicts arise from this scenario. Thus, cases with $n = 2, 3, 4$ agents were tested.

Environment dimensions	$(540, 540)$
Number of agents n	$\{2, 3, 4\}$
Max nodes η_{max}	4000
Min nodes η_{min}	$\eta_{max}/5$
Reducing rate σ	10
Min angle α_{min}	90
Smoothing s	0
Radius r	5

Table 3.2: Parameters of experiment 3.4.2 concerning parameter n

Parameter setting In this experiment, a similar setting as in the experiment 3.4.1 is used, with the difference being a pattern in which agents should move. Their individual optimal solutions are arranged in a star-like manner, so that they should collide at once in the centre of the map. Their initial and goal positions are coordinates evenly spaced on a circle, with its centre being the colliding position through which all agents should pass.

For this experiment, three different values of n were tested, with 10 iterations running for each of them.



(a) A case with $n = 3$ (b) A case with $n = 4$

Figure 3.4: Output examples for two different values of n. Agents' paths are arranged in a star-like pattern to provide one common conflict spot.

Hypothesis In the previous experiment, since paths were arranged in a grid-like pattern, only pairs of agents were to collide at one time at one position. This experiment sets to make them collide all at once. The hypothesis is that the average path lengths will overall be higher than in the previous experiment, as agents will need to avoid more collisions at once, therefore straying them further from their optimal path.

Results Figure 3.5 shows that in case of $n = 4$, scatter between reached values was significantly smaller than in the other cases. Although cases with $n = 3$ were able to reach shorter path lengths, the smallest average of average path lengths was reached in $n = 4$. Examples of two test runs for $n = 3$ and $n = 4$ can be seen in figure 3.16.

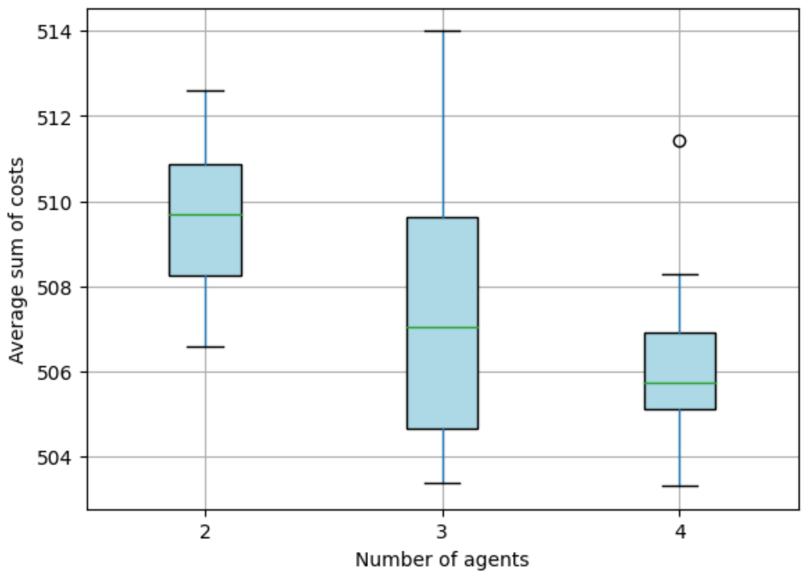


Figure 3.5: Boxplot graph depicting results of experiment 3.4.2. Each box is for a different number of agents, with y axis representing average path costs for one agent.

Interpretation One possible explanation for the overall success of $n = 4$ could possibly be attributed to the space pruning caused by CE-CBS. This algorithm, by adding constraints prohibits RRT* nodes to be created at positions which would break constraints. They therefore explore more of the remaining space, allowing the path to be more optimised, as if a large number of constraints is present, a larger area is restricted and not searched through. The more agents present in a test scenario, the more conflicts appeared, creating more constraints.

3.4.3 Maximal number of RRT* nodes η_{max} - empty space

Parameter η_{max} is essential not only for the computational time of the algorithm, but most importantly for increasing the path quality. The more RRT* is allowed to search the environment, the more it optimises its paths and approaches convergence to the optimal solution. As running RRT* without a constrained value of η_{max} could lead to potentially infinite computational time, it is necessary to compromise between computational time and path quality. This and the following experiments are set to determine which value of η_{max} maintains a realistic computational time, represented here by the number of CE-CBS iterations on which it is directly dependent, while keeping the paths as short as possible.

Parameter setting The agent setting is the same as in experiment 3.4.2 with $n = 3$, as 3 agents are also used in this experiment. 8 values of η_{max} are

Environment dimensions	$(540, 540)$
Number of agents n	3
Max nodes η_{max}	$\{1000, 2000, ..., 8000\}$
Min nodes η_{min}	$\eta_{max}/5$
Reducing rate σ	10
Min angle α_{min}	90
Smoothing s	0
Radius r	5

Table 3.3: Parameters of experiment 3.4.3 concerning parameter η_{max}

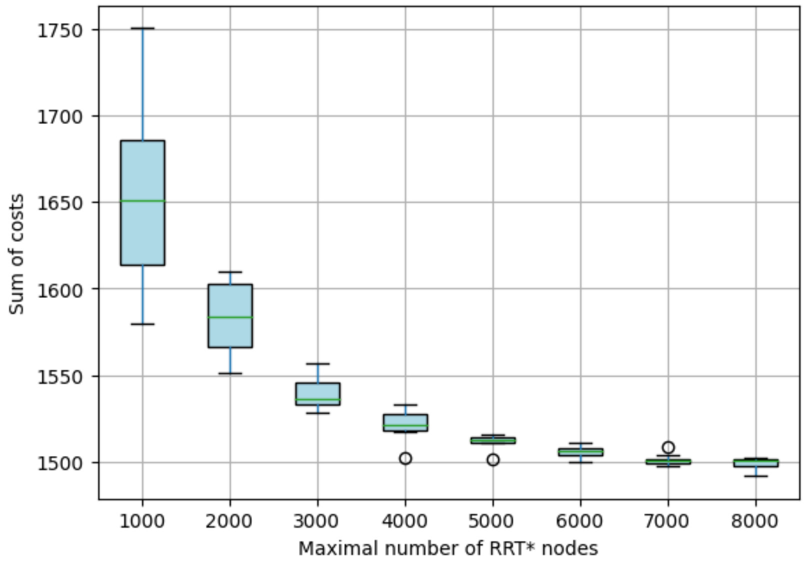
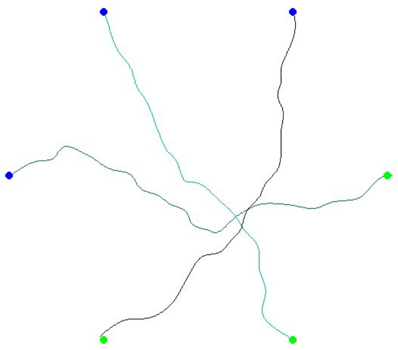


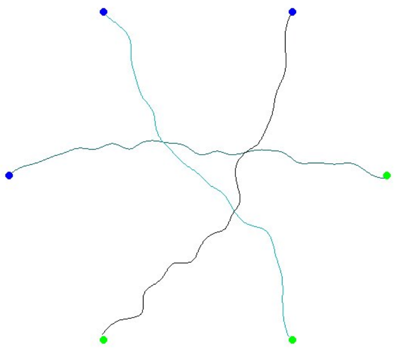
Figure 3.6: Figure depicting results of experiment 3.4.3 showing relationship between average path length per agent and η_{max}.

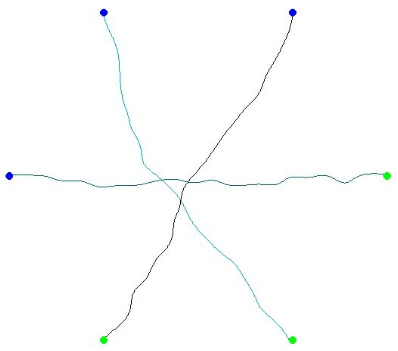
tested, with 10 iterations per value.

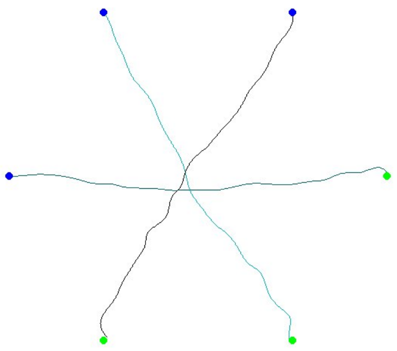
Hypothesis The RRT* algorithm is proved to converge to an optimal solution with $\eta_{max} = \infty$ [Karaman and Frazzoli, 2011]. However, with constraints in place, the paths should be longer than the optimal single-agent solution to avoid collision, which is set to happen in the middle of the map. Therefore, with increasing η_{max}, the paths should generally become smoother. It is also worth studying how the number of conflicts that arise will change depending on η_{max}. As lower values should result in paths prone to detours, collisions may not occur at all, whereas paths closer to optimal solutions could provoke more conflicts to arise.

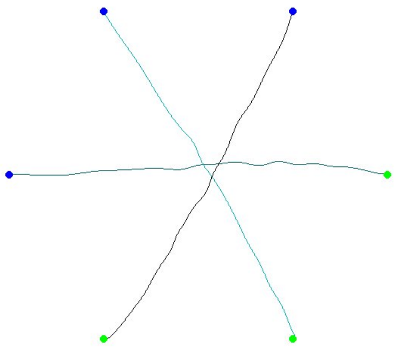
Results As can be seen in Figure 3.6, sum of costs (y axis) decreases with increasing η_{max}. The scatter between values also decreases with increasing η_{max}. These changes become less apparent after $\eta_{max} \geq 5000$. The opposite


(a) A case with $\eta_{max} = 1000$


(b) A case with $\eta_{max} = 2000$


(c) A case with $\eta_{max} = 3000$


(d) A case with $\eta_{max} = 4000$


(e) A case with $\eta_{max} = 6000$

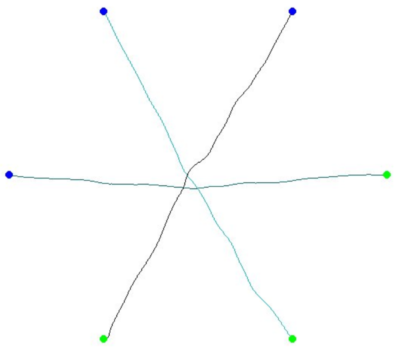

(f) A case with $\eta_{max} = 8000$

Figure 3.7: Output examples for six different values of η_{max}. Agent paths are arranged in a star-like pattern as in experiment 3.4.2, with 3 agents being present in the environment.

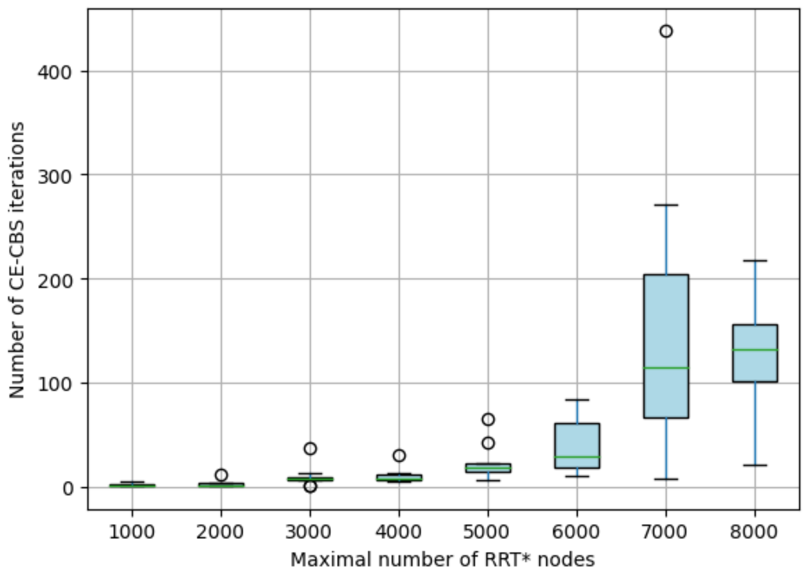


Figure 3.8: Figure showing how the number of CE-CBS iterations changed based on different η_{max}.

appears to be true for the number of CE-CBS iterations, which means that the number of CE-CBS nodes expanded during the course of the algorithm. The dependence of CE-CBS iterations on η_{max} is shown in Figure 3.8. As η_{max} increases, so does the number of iterations, also with higher scatter. This becomes more apparent with $\eta_{max} \geq 5000$, with a large jump in both the values reached and the scatter in $\eta_{max} = 7000$. Examples of six outputs for six values of η_{max} are shown in Figure 3.22

Interpretation As higher values allow RRT* to explore the environment more, it is able to provide a more path closer to the optimal path. With paths approaching optimum however more often come situations where collisions arise, as the setting of this experiment suggests. With lower values of η_{max}, agents did not collide as often because of frequent detours they take. With more conflicts appearing, the space becomes more constrained and allows RRT* to also explore non-constrained parts more.

3.4.4 Maximal number of RRT* nodes η_{max} - large spaces between obstacles

This experiment focuses on parameter η_{max} as well as experiment 3.4.3. This experiment, however, deals with an environment with several obstacles, with two agents starting at each other's goal locations. This means that their optimal individual path should be the same, only reversed. However, this is impossible, as they would collide in the middle. CE-CBS has to ensure that no collisions happen, and thus at least one agent's path has to be detoured to avoid the other agent. Collisions occur when two paths intersect and both agents arrive at the intersection at a similar time, denoted by unsafe intervals.

Parameter setting Two agents are used in this scenario, with three obstacles between them. Ten values of η_{max} are tested, with 10 iterations per value.

Environment dimensions	$(550, 300)$
Number of agents n	2
Max nodes η_{max}	$\{1000, 2000, ..., 10000\}$
Min nodes η_{min}	$\eta_{max}/5$
Reducing rate σ	10
Min angle α_{min}	90
Smoothing s	0
Radius r	5

Table 3.4: Parameters of experiment 3.4.4 concerning parameter η_{max}

Hypothesis As in 3.4.3, path lengths should decrease with increasing value of η_{max}. The scatter between them should decrease too, as when higher η_{max} will be set, larger space will be searched, decreasing the occurrence of detours.

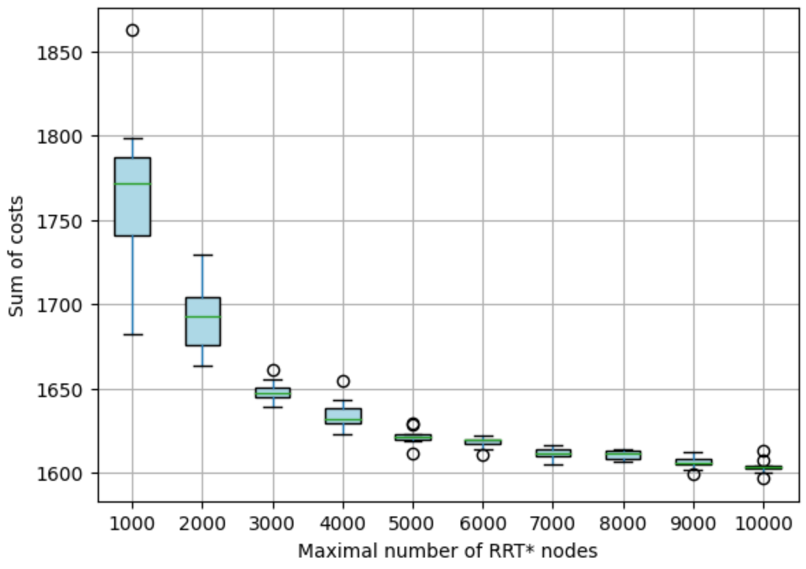
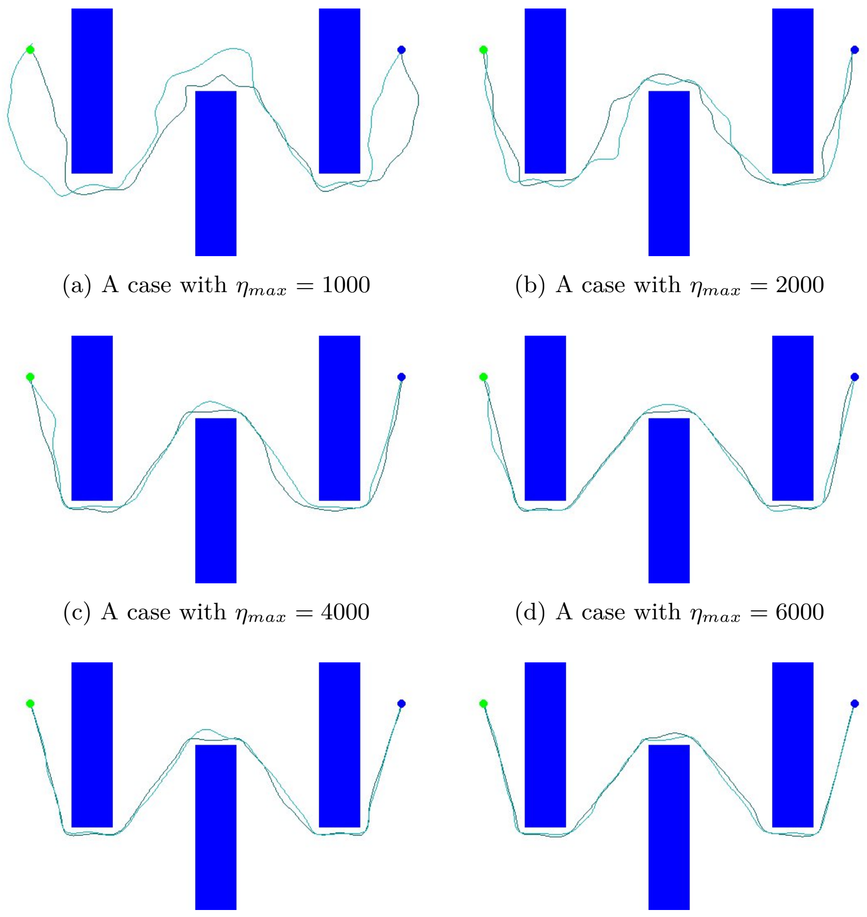


Figure 3.9: Boxplot graph depicting results of experiment 3.4.4. Each box represents sums of costs depending on values of η_{max}.

Results Figure 3.9 shows that with increasing value of η_{max} decrease both the average path length and the scatter between individual results for each value. The most apparent differences are in the range $1000 \leq \eta_{max} \leq 5000$, after which the decrease in path lengths slows down.



(a) A case with $\eta_{max} = 1000$ (b) A case with $\eta_{max} = 2000$

(c) A case with $\eta_{max} = 4000$ (d) A case with $\eta_{max} = 6000$

(e) A case with $\eta_{max} = 8000$ (f) A case with $\eta_{max} = 10000$

Figure 3.10: Output examples for six different values of η_{max}.

Interpretation As the two agents approach each other, conflicts appear only if they are close to each other in both position and time. With larger η_{max}, paths of both agents become more similar, as in an optimal case, and when solving only a single-agent scenario, those paths should be the same. This can be seen in figure 3.10. The two agents can meet in the central position on the map. to ensure they avoid, paths take different directions around this position.

3.4.5 Agents on the same path - changing radius

This experiment visualises a scenario in which two agents would optimally share a path, a starting node of one agent being the goal node of the second agent and vice versa. As agents should collide along the middle of their paths, they would have to navigate to avoid this area, but in a way which does not create another area of conflict, meaning the two agents would not be able to dodge in the same direction. Agents have to take into account their bodies and plan a path so that they don't overlap at any time. This experiment studies different sizes of agent bodies to show how paths change depending on this parameter.

Parameter setting Increasing sizes of agent bodies represented by the radius parameter r were tested. Ten iterations were performed for each tested value. Several obstacles were present in the environment and agents were placed in a way which made starting position of one agent the goal position of the other agent. This was to make them plan a path that avoids overlapping with the other agent body.

Environment dimensions	$(550, 300)$
Number of agents n	$\{2, 4, 6, 8, 10\}$
Max nodes η_{max}	4000
Min nodes η_{min}	$\eta_{max}/5$
Reducing rate σ	10
Min angle α_{min}	90
Smoothing s	0
Agent radius r	$\{5, 6, ..., 10\}$

Table 3.5: Parameters of experiment 3.4.5 concerning parameter r

Hypothesis Increasing the radius r should make it more difficult for the agents to not overlap at any time. They should plan longer detours, the larger they are to accommodate their bodies. This should lead to increased path lengths with higher values of r.

Results Sum of costs increased with increasing r without any exceptions. The scatter between values decreased slightly.

Interpretation As radius r increased, agents did in fact plan longer detours around the part of the environment where they would meet to accommodate their sizes. The slight decrease in scatter could be attributed to the fact that with larger bodies, agents have to both maintain larger distance from static obstacles and set larger unsafe intervals, thus pruning the search space more with increasing r.

3.4.6 Smoothing parameter s - empty space

This experiments tests the minimal value of the smoothing parameter s. The higher its value, the smoother the path, although it is farther from the original path planned by RRT*. This does not necessarily have to be a disadvantage, as in all scenarios tested in this work, a constrained value of maximal number of RRT* nodes is set. This may keep the algorithm from converging to an optimal solution, and additional path smoothing could potentially help shorten these sub-optimal paths.

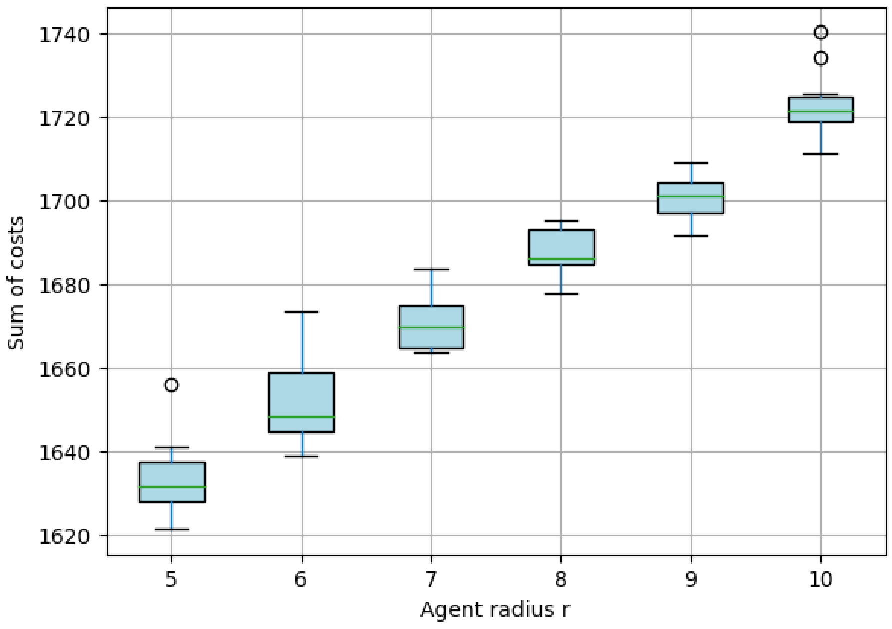
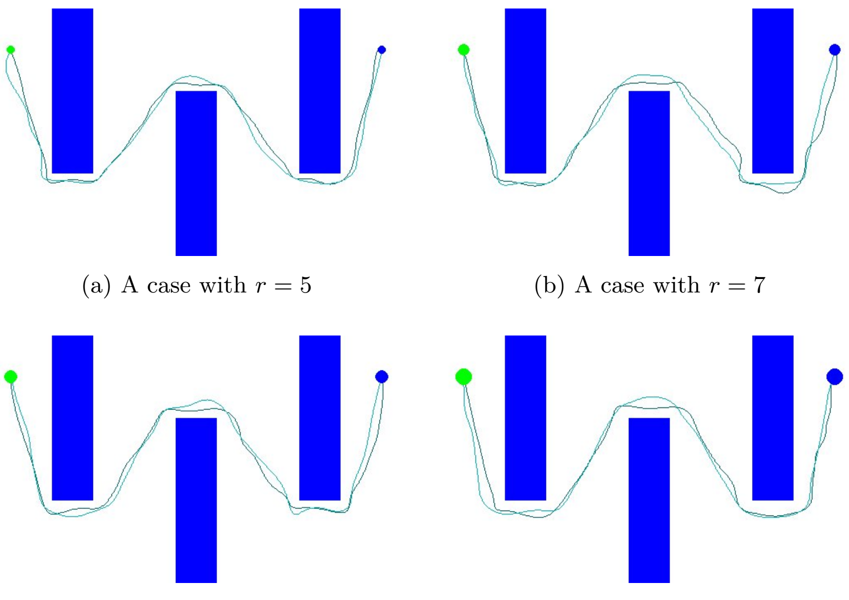


Figure 3.11: Results of experiment 3.4.5 with increasing value of agent radius r.



(a) A case with $r = 5$

(b) A case with $r = 7$

(c) A case with $r = 8$

(d) A case with $r = 10$

Figure 3.12: Output examples for four different values of r.

Environment dimensions	$(540, 540)$
Number of agents n	3
Max nodes η_{max}	3000
Min nodes η_{min}	$\eta_{max}/5$
Reducing rate σ	10
Min angle	90
Smoothing s	$\{0, 1, ..., 4\}$
Radius r	5

Table 3.6: Parameters of experiment 3.4.6 concerning parameter s.

Parameter setting For this experiment, the lower value of $\eta_{max} = 3000$ was purposefully used to illustrate how increasing s will manage paths prone to frequent detours. Ten iterations were performed for each value of s. Agents were once again set to move in a star-shaped manner, forcing them to collide were they to plan a path converging to optimum.

Hypothesis It is possible that combining higher value of s and lower value of η_{max} may yield similar results in terms of path quality as cases studied previously, where higher η_{max} and $s = 0$ values were used. This means that with increasing s, average path lengths should be shorter.

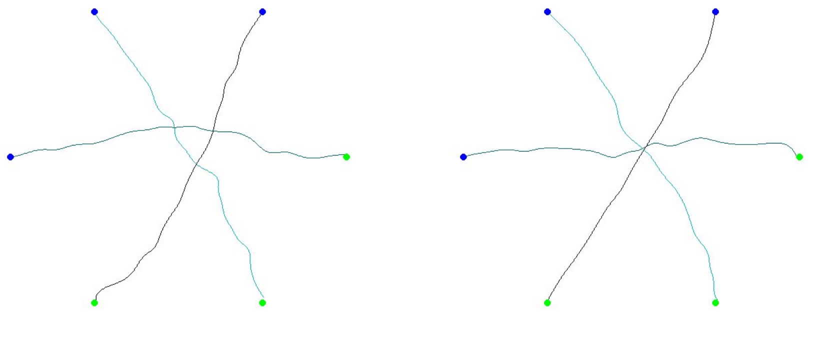

(a) A case with $s = 0$ (b) A case with $s = 4$

Figure 3.13: Output examples for two different values of s. Agents' paths are arranged in a star-like pattern to provide one common conflict spot.

Results Highest scatter is present in $s = 0$, with averages of average sum of costs decreasing with increasing s, with the exception of $s = 0$. Both the lowest reached average sum of costs and average of these values was reached in $s = 4$.

Interpretation Although changes in path lengths between different values of s are not as high as for example changes in values of η_{max}, results suggest that increasing s does shorten paths. It is though necessary to point out that this is in an environment without static obstacles which may require paths to

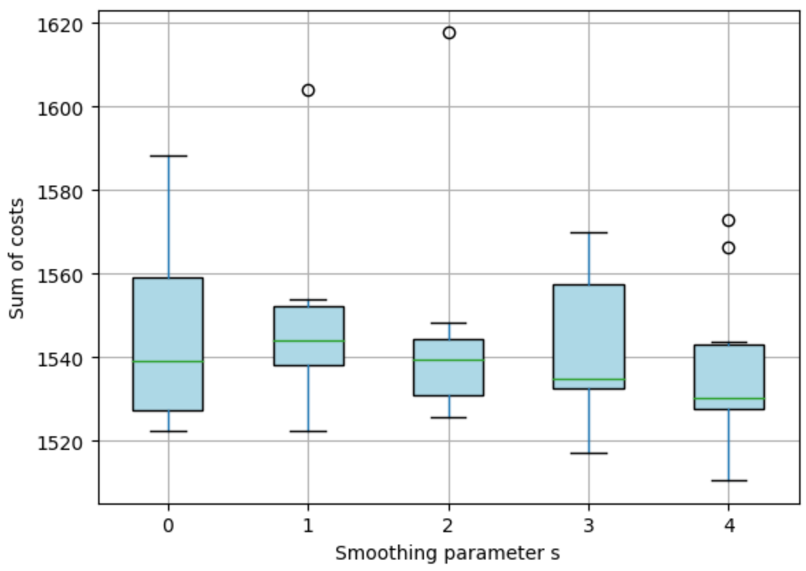


Figure 3.14: Boxplot graph depicting results of experiment 3.4.6.

contain several turns. In those environments, smoothing could potentially lead to increased collisions with static environment. These cases are studied below.

3.4.7 Smoothing parameter s – large obstacles

Parameter setting Parameters in this experiment are set the same as in experiment 3.4.6, but using a map with several obstacles in which two agents move in direction against each other, start position of each agent being the goal position of the other agent.

Environment dimensions	$(550, 300)$
Number of agents n	3
Max nodes η_{max}	3000
Min nodes η_{min}	$\eta_{max}/5$
Reducing rate σ	10
Min angle	90
Smoothing s	$\{0, 1, ..., 4\}$
Radius r	5

Table 3.7: Parameters of experiment 3.4.7 concerning parameter s

Hypothesis Experiment 3.4.6 has shown that increasing s might shorten the paths. Environment used in this experiment however includes several static obstacles, which may influence how the path quality evolves.

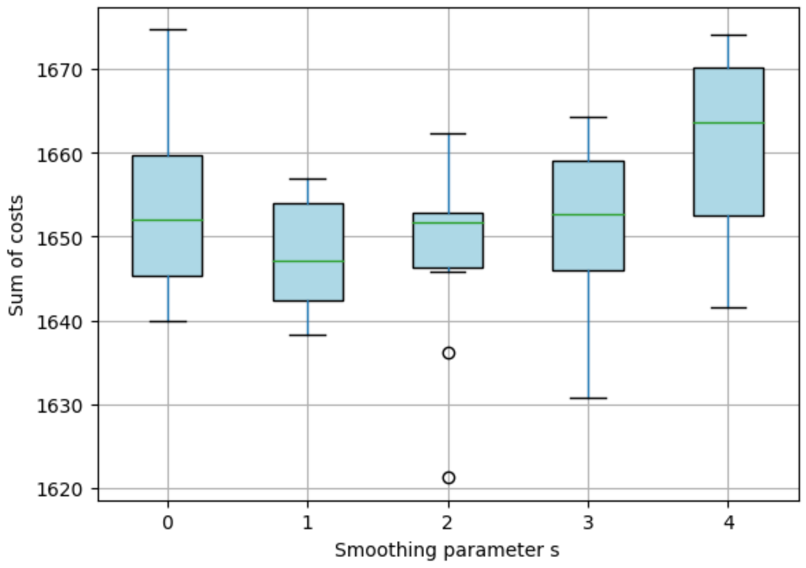
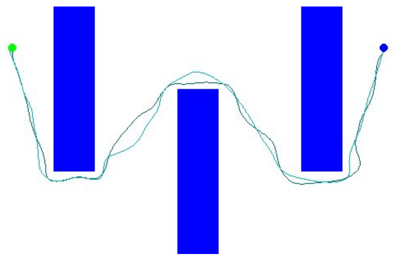


Figure 3.15: Boxplot graph depicting results of experiment 3.4.7 concerning parameter s.


(a) A case with $s = 0$

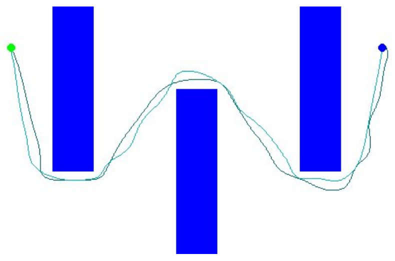

(b) A case with $s = 4$

Figure 3.16: Output examples for two different values of s. Agents move in direction against each other in an environment with several obstacles.

Results The sum of costs on average decreased between $s = 0$ and $s = 1$, but after that it continued to increase again. The least amount of scatter between values is in $s = 2$, but is apparent in all other values tested.

Interpretation Contrary to experiment 3.4.6, path lengths showed rising trend in this experiment with the exception of $s = 0$ having higher average sum of costs than $s = 1$ and $s = 2$. This may mean that while some values of $s > 0$ may be suitable for this setting, but unlike in a setting with no obstacles high values of s do not yield better results. Therefore it might be better to refrain from using higher values of s in scenarios with static obstacles present.

3.4.8 Increasing minimal angle α_{min} - 2 agents

The parameter α_{min} signifies the lowest possible angle between the two neighbouring path segments. The higher it is, the less sharp turns there should be on the path.

Environment dimensions	$(540, 540)$
Number of agents n	2
Max nodes η_{max}	10000
Min nodes η_{min}	η_{max} / 5
Reducing rate σ	10
Min angle α_{min}	$\{80, 85, ...120\}$
Smoothing s	0
Radius r	5

Table 3.8: Parameters of experiment 3.4.8 concerning parameter α_{min}.

Parameter setting In this scenario, two agents were used. Nine different values of α_{min} were tested, with 10 iterations run per instance. The base value of α_{min} was set as $\alpha_{min} = 90°$ in other experiments. Agents move in a maze map with several low-width obstacles. This environment should force agents to plan a path containing sharp turns, should the value of α_{min} allow them to.

Hypothesis The sum of costs could increase with increasing α_{min}, as with lower values, agents may use sharper turns around the narrow obstacles to traverse a shorter path.

Results Figure 3.20 shows, that between values of 80 and 110, both sum of costs and scatter have oscillated, providing no definite result. However, between the values of $\alpha_{min} \in \{110, 115, 120\}$, the sum of costs has notably risen, with the scatter in value $\alpha_{min} = 120°$ being higher than the lower values.

Interpretation The results suggest that up until a certain value, parameter α_{min} could have little influence over the path length. This could be beneficial for allowing the movement to be more constrained, with agent using less sharp turns. However, as a sharp path cost rise can be noted in $\alpha_{min} > 110$, an upper bound could be set should path lengths and variability between reached values get to significantly higher values, as is the case with $\alpha_{min} = 120°$.

This experiment was the first where more iterations of RRT* had to be run because the algorithm was unable to adjust an angle. In these cases, η_{max} is lowered to produce a less optimal result, which on the other hand might bypass an obstacle from a greater distance, avoiding the sharp angle problem. This was discussed in Section 2.2.1.1. However, as these experiments have shown, mainly due to values of η_{min}, faulty cases were produced in some runs where α_{min} was set to large values such as $\alpha_{min} = 120°$. As the iterations were lowered too much and too quickly, no valid path was found before η_{max} was lowered to a value that made it impossible for RRT* to find the goal position. To combat this, η_{min} can be increased so that a valid result is guaranteed. Another way is to slow down the pace by which η_{max} is lowered. This was

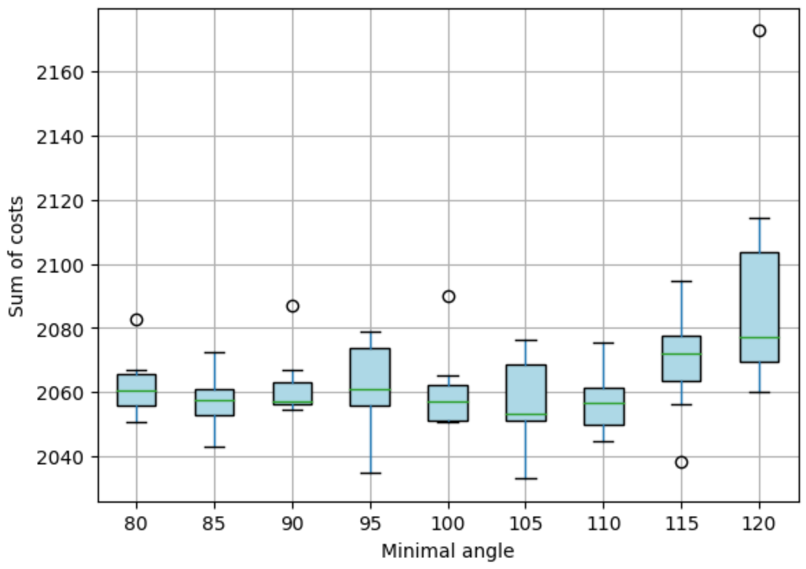


Figure 3.17: Boxplot graph depicting results of experiment 3.4.8 concerning parameter α_{min}.

previously a static value, and the new η_{max} was set as $\eta_{max} = \eta_{max} - \frac{\eta_{max}}{10}$. A new parameter *reducing rate* σ is introduced instead of the denominator 10 to make the tuning of this process possible.

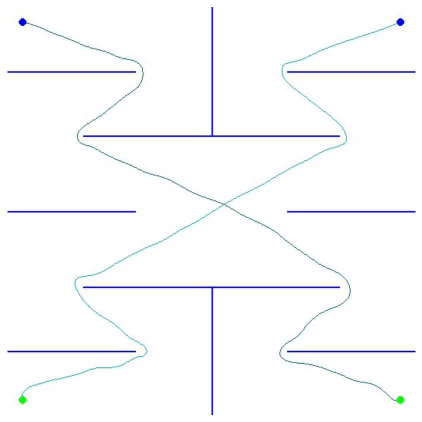
With this, it may still be possible for RRT* to return a path that breaks an angle constraint if it is still unable to adjust all angles and α_{min} is set high. As these results appeared mainly in cases with $\alpha_{min} \geq 120°$, it sets an upper bound to the range of α_{min}.

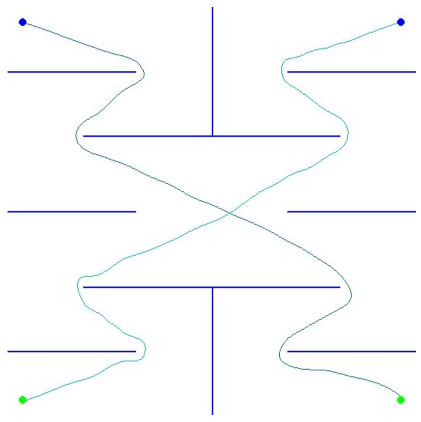
To ensure that the angle conditions are met every time, a modification was made to the lower level of the CE-CBS function, originally shown in the algorithm 5. Now after lowering η_{max} the maximal amount, RRT* would not return a final result, but iterate again until it finds a suitable solution. This only influences the cases where a solution would not be found otherwise.

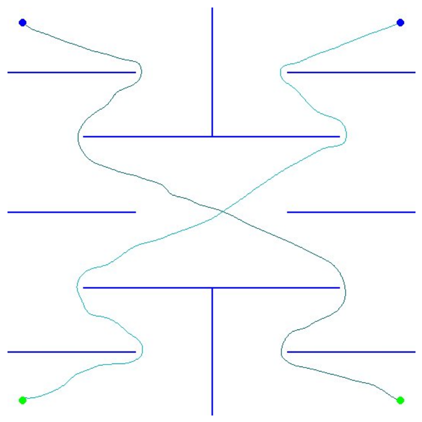
Figure 3.19 presents two examples of a single-agent run, where in case of 3.19a angles were adjusted successfully, but in case of 3.19b η_{max} was lowered too much and no solution was found.

3.4.9 Increasing minimal angle α_{min} - 4 agents

This experiment follows modifications suggested in experiment 3.4.8, where in response to an emerging issue causing RRT* searching too small of a portion of the environment, algorithm 6 was modified to ensure that the space is searched properly until a solution is found. This experiment is the first to use this setting along with the user modifiable parameter σ, which controls how fast η_{max} decreases between iterations if a solution is not found.


(a) A case with $\alpha_{min} = 80°$


(b) A case with $\alpha_{min} = 95°$


(c) A case with $\alpha_{min} = 110°$

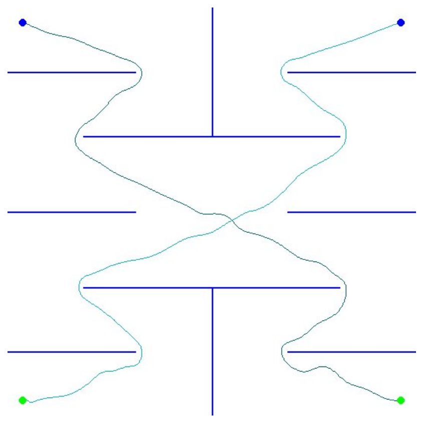
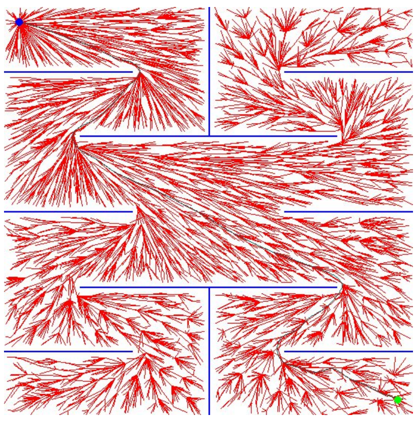

(d) A case with $\alpha_{min} = 120°$

Figure 3.18: Output examples for four different values of α_{min}.

Parameter setting The experiment 3.4.8 suggested an upper bound of $\alpha_{min} = 120°$, in which the algorithm had problems in achieving a viable solution. As this was resolved by modifying the algorithm 6, the maximum value of α_{min} studied here was raised to $\alpha_{min} = 125°$.

Four agents are present at the start in each corner of the map, and for each of these agents, their goal positions lie on a diagonal in an opposite corner of the map. For example, if an agent's starting position is placed at $(20, 20)$, its goal position is $(520, 520)$. This means that two pairs of agents arise, where one agent's start position is the other agent's goal position, and vice versa.

Hypothesis Based on results of experiment 3.4.8, the values here could oscillate around the same value and the scatter range, with an increase around $\alpha_{min} = 120°$. However, a pair of agents will also have to avoid themselves when traversing a similar path in opposite directions, thus possibly altering the result.


(a) An example where α_{min} condition was successfully met.

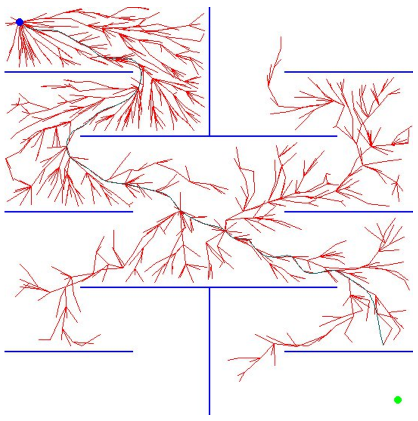

(b) An example where a solution was not found.

Figure 3.19: Output examples for cases with one agent, $\eta_{max} = 10000$ and $\alpha_{min} = 125°$. The black line represents the paths found; the red lines connect the vertices of the RRT* tree.

Environment dimensions	$(540, 540)$
Number of agents n	4
Max nodes η_{max}	6000
Min nodes η_{min}	η_{max} / 2
Reducing rate σ	20
Min angle α_{min}	$\{80, 85, ...125\}$
Smoothing s	0
Radius r	5

Table 3.9: Parameters of experiment 3.4.9 concerning parameter α_{min}.

Results The highest sums of costs were reached in $\alpha_{min} = 80°$, where the overall average of these values was the highest. In addition, the lowest scatter was observed in this case, with values being progressively more spaced out, with the exception of $\alpha_{min} = 105°$.

Interpretation Unlike experiment 3.4.8, the values here do not show a definite increase in the sum of costs and show a higher scatter and a wider range of values reached. This can be attributed to the two pairs of agents on the same path, which may have a higher impact on path costs and provide a higher variety of results. Despite this, it seems that except for values such as $\alpha_{min} = 105°$, the averages of the sum of costs have a mostly decreasing trend, with scatter slightly increasing with increasing α_{min}.

3.4.10 Path planning in narrow crossing space

This experiment is to test path planning in a setting where a conflict is probable and difficult to avoid. This is represented by a map with two perpendicular narrow passages crossing in the middle. Two agents are present in this experiment, one traversing each narrow passage. If they were to plan

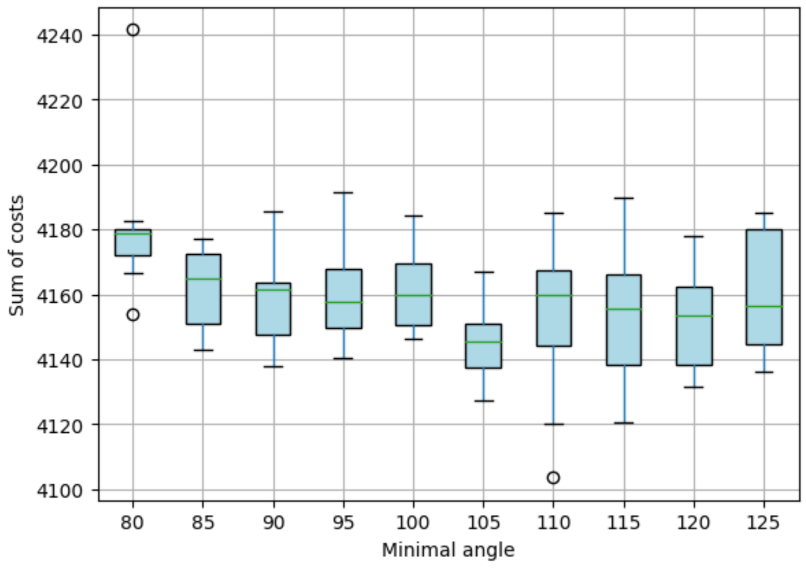


Figure 3.20: Results of experiment 3.4.9 concerning parameter α_{min}.

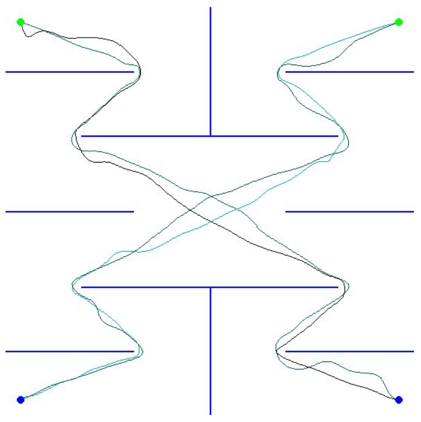
an optimal path that did not take the other agent into account, they would collide in the middle of the map. As there are limited possibilities of collision avoidance, this experiment aims to study the model's behaviour in this type of environment.

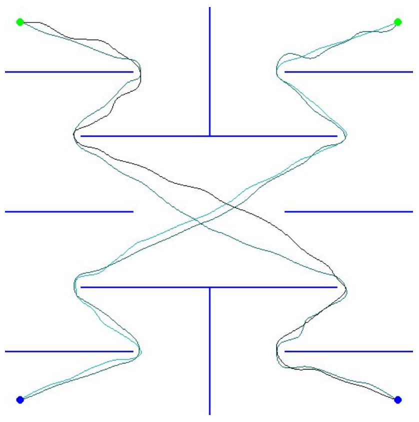
Parameter setting Two agents are moving in two narrow hallways crossing in the centre of the environment. Six different values of η_{max} are tested here to see how well they can avoid each other based on the quality of their individual paths. Ten iterations were run for each instance.

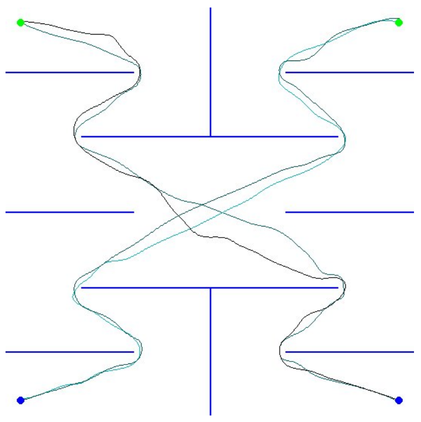
Environment dimensions	$(240, 240)$
Number of agents n	2
Max nodes η_{max}	$150, 200, ..., 400$
Min nodes η_{min}	η_{max} / 2
Reducing rate σ	20
Min angle	α_{min} 90
Smoothing s	0
Radius r	5

Table 3.10: Parameters of experiment 3.4.10.

Hypothesis As previous experiments have shown, the sum of costs tends to decrease with increasing η_{max}, which can also lead to an increased amount of collisions. It is therefore necessary to find a middle ground between path quality and computational load. If several hundreds of CE-CBS iterations were necessary, that iteration would take hours to complete.


(a) A case with $\alpha_{min} = 80°$


(b) A case with $\alpha_{min} = 95°$


(c) A case with $\alpha_{min} = 110°$

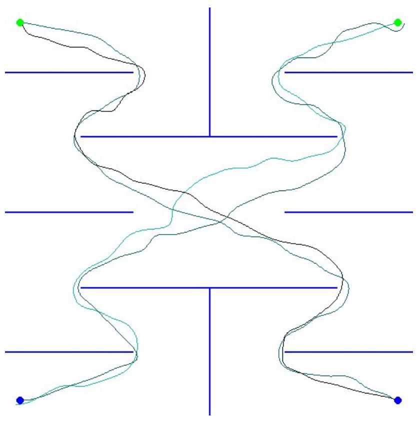
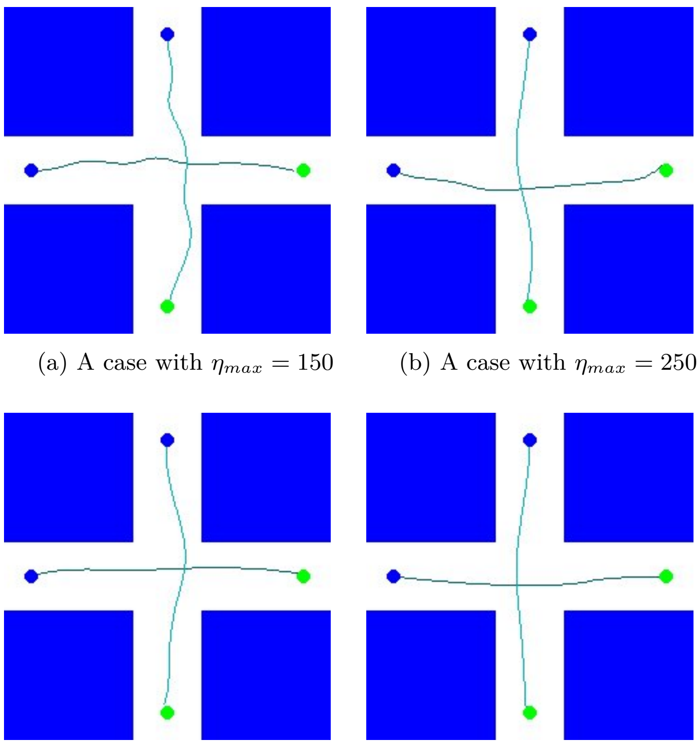

(d) A case with $\alpha_{min} = 125°$

Figure 3.21: Output examples for four different values of α_{min} in experiment 3.4.9 with four agents.

Results Figure 3.23 shows that with increasing η_{max}, the sum of costs first increases from $\eta_{max} = 150$ to $\eta_{max} = 200$, then begins to decrease. However, these changes are only in the matter of units. Scatter is also significantly lower with higher η_{max}, starting with $\eta_{max} = 250$. Figure 3.24 shows the number of CE-CBS iterations performed for instances with different η_{max}. In case of values from $\eta_{max} = 150$ to $\eta_{max} = 300$, the whole computation would take tens of seconds to several minutes, as the number of iterations was lower. However, from $\eta_{max} = 350$, the number of CE-CBS iterations increased in some cases to hundreds of iterations, making the computation take over eight hours in the outlaying case.

Interpretation From these results it seems that an upper bound must be set to η_{max} to prevent long computational times. The optimal value should depend on the environment. Here, from figure 3.24 it seem that η_{max} should be no more than 300, while from 3.23, it seems that better, more consistent results are reached with $\eta_{max} \geq 250$. Therefore, the best value of η_{max} in this


(a) A case with $\eta_{max} = 150$ (b) A case with $\eta_{max} = 250$

(c) A case with $\eta_{max} = 300$ (d) A case with $\eta_{max} = 400$

Figure 3.22: Examples of four cases with different η_{max}.

environment could be $\eta_{max} \in [250, 300]$.

3.4.11 Agents on the same path – hall map

In this experiment, two agents are placed against each other in an obstacle-free path, a narrow hallway. These agents should switch positions, meaning that one agent's start position is the other agent's goal position. As previous experiments have shown, this is easily achievable. This experiment will focus more on the computational load than on the sum of costs itself. The focus is on the dimension marking the width of the hall, which denotes how much space the two agents have to avoid each other. The lower it is, the closer they have to get to each other. The goal of this experiment is to find out how the number of CE-CBS iterations which directly influences the computational time changes based on the amount of free space agents that get to have around them.

Parameter setting Previous experiments have established that if the dimensions of the environment change, so must η_{max}, to keep balance between the sum of costs and the number of CE-CBS iterations, and thus the computational time. To get an idea of what these values may be in the case of the narrow hall environment, a preliminary experiment was performed on the case where the width of the environment was set to $y = 200$. The results of

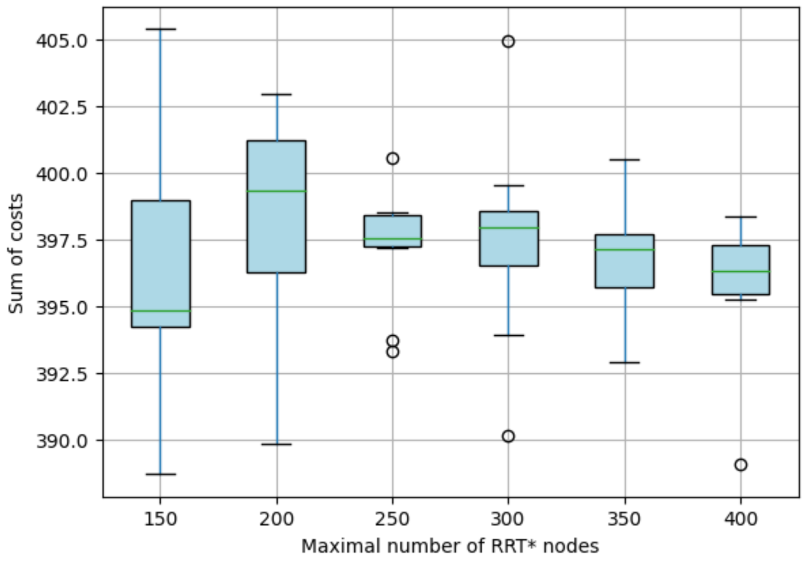


Figure 3.23: Result of experiment 3.4.10 depicting the relationship between the sum of costs and η_{max}.

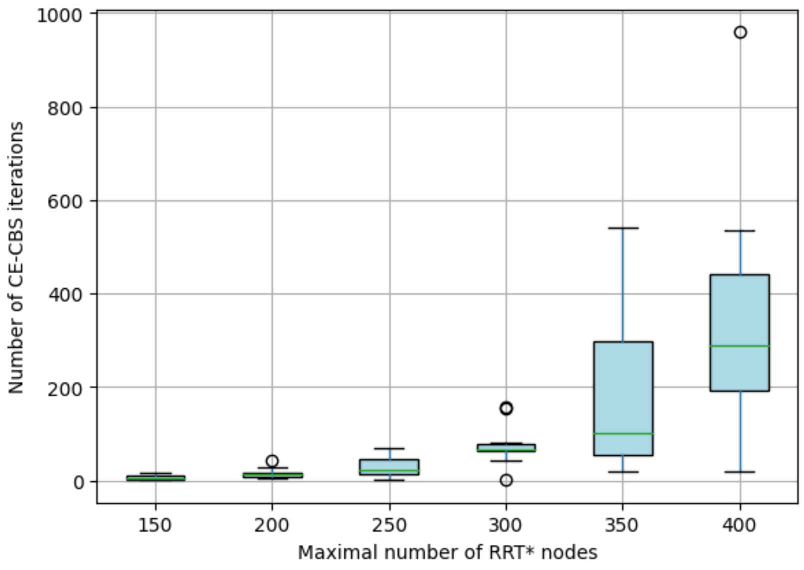


Figure 3.24: Result of experiment 3.4.10 depicting the relationship between number of CE-CBS iterations and η_{max}.

this experiment can be seen in figures 3.25 and 3.26. It can be seen that the sum of costs decreases with increasing η_{max}, but these differences are mainly

Environment dimensions	$(200, \{25, 50, ..., 200\})$
Number of agents n	4
Max nodes η_{max}	$\{750, 1000, ..., 2500\}$
Min nodes η_{min}	$\eta_{max}/2$
Reducing rate σ	20
Min angle α_{min}	90
Smoothing s	0
Radius r	5

Table 3.11: Parameters of experiment 3.4.11

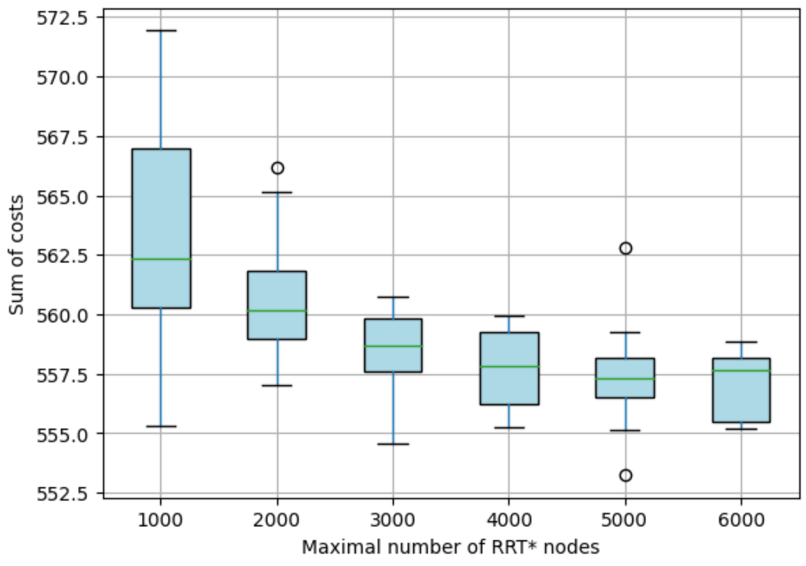


Figure 3.25: Result of preliminary experiment for experiment 3.4.11 depicting the relationship between the sum of costs and η_{max}.

in units. For the number of CE-CBS iterations, this value sharply increases with $\eta_{max} \geq 3000$. Due to this, the value of η_{max} was set as $\eta_{max} = 2500$ for the highest value studied of $y = 200$. For lower values, η_{max} was chosen to decrease by 10 units for each point of width y. This means that for the lowest value of $y = 25$, $\eta_{max} = 750$.

Hypothesis As η_{max} is adjusted for every y, both the sums of costs and the iterations of CE-CBS should reach similar values for different y. If this is not the case, then the relationship between η_{max} and y should be defined differently to better represent the different search scenarios.

Results Figure 3.27 shows the values reached for the sum of costs. It can be seen that the values move mostly in the range of $(555, 560)$. This is the case for all values of y without severe outliers. On the other hand, figure 3.28, depicting

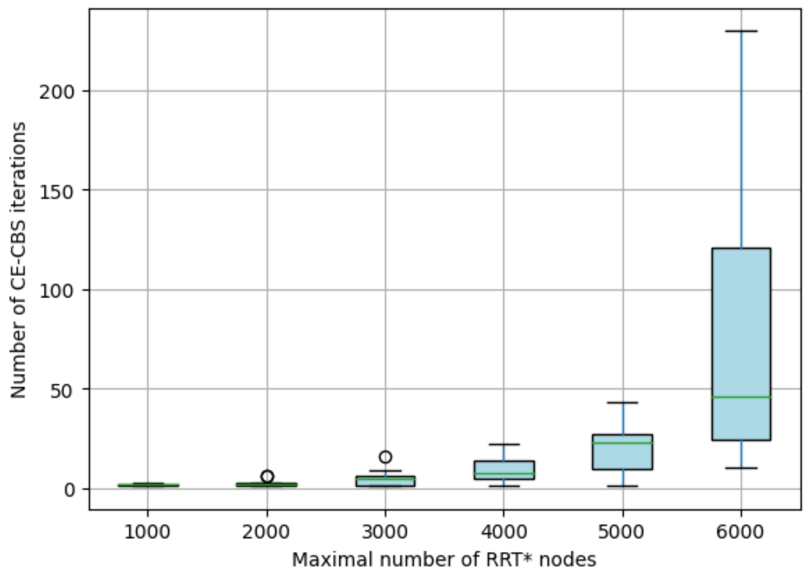


Figure 3.26: Result of preliminary experiment for experiment 3.4.11 depicting the relationship between number of CE-CBS iterations and η_{max}.

the number of CE-CBS iterations shows that with $y = 25$, significantly higher values and scatter were reached. Example runs are shown in figure 3.29.

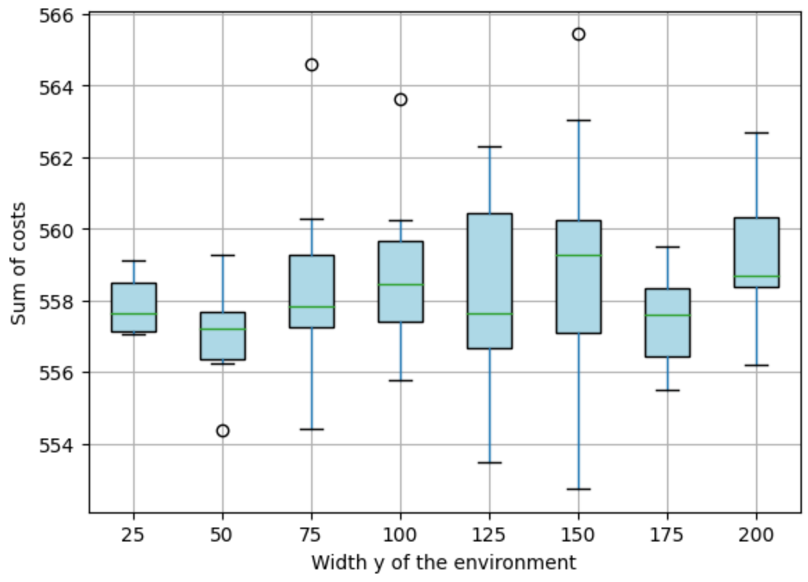


Figure 3.27: Resulting sum of costs of experiment 3.4.11.

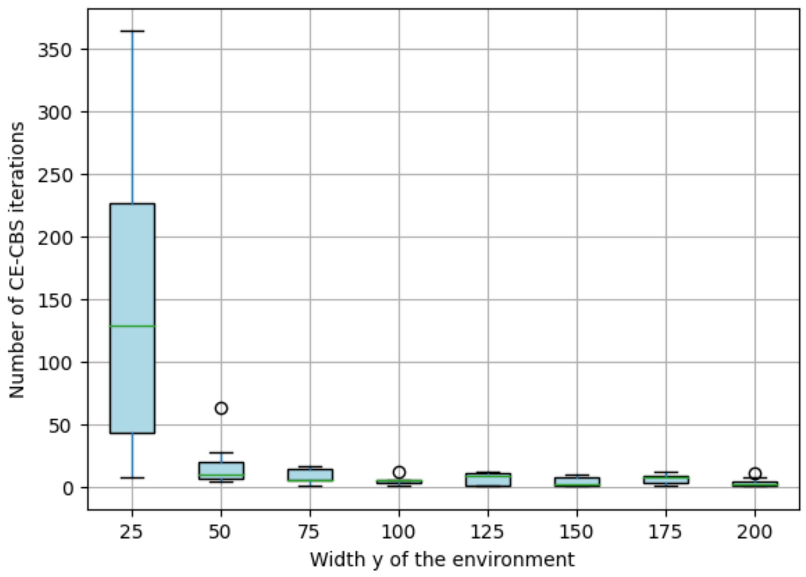
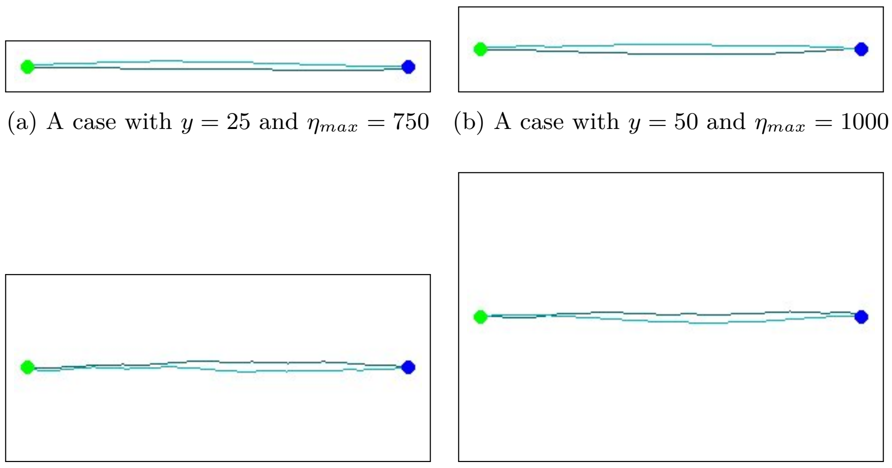


Figure 3.28: Resulting numbers of CE-CBS iterations of experiment 3.4.11.



(a) A case with $y = 25$ and $\eta_{max} = 750$ (b) A case with $y = 50$ and $\eta_{max} = 1000$

(c) A case with $y = 125$ and $\eta_{max} = 1750$ (d) A case with $y = 200$ and $\eta_{max} = 2500$

Figure 3.29: Examples of four cases with different width of the hallway y.

Interpretation Due to the difference between the number of CE-CBS iterations in the case of $y = 25$ and other values studied, it seems that the chosen $\eta_{max} = 750$ for this case was set too high, as it is the only case with high computational time, as in other cases, the number of CE-CBS iterations was mostly kept below 10. This suggests that η_{max} was chosen appropriately for these cases, as this value is very low and the sums of costs themselves vary only by units. Another possible cause of the outlaying $y = 25$ is the fact that, especially in this most narrow environment, agents have little space to avoid each other, as they both have a diameter of 10. This makes it more likely for them to collide, although they do not seem to share this issue in $y = 50$, which is the second most narrow map. These results suggest that it is indeed necessary to have η_{max} depend on the size of the environment to obtain consistent results both in path costs and computational time.

3.5 Discussion

The experiments have shown that the model can adapt to several different types of environment and that the parameters of the model can be adjusted to solve a specific scenario. It was shown that of the parameters studied, the parameter of the maximum number of RRT * nodes η_{max} had the highest impact on the quality of the solution. The computational time is tied to the number of CE-CBS iterations, so when more conflicts arise while planning, the computational load is higher, as RRT* is re-planned several times. The duration of this computation is also influenced by η_{max}. It was shown that in cases with a higher number of collisions, η_{max} can be supplemented to some extent, as collisions prune the search space and allow RRT* to search more of the unconstrained part of the environment.

Based on the results of the experiments, it seems that agents are able to navigate both narrow hallways and spaces where multiple conflicts can occur at once without colliding, while still searching for the shortest path. The agents can move on smooth curves while satisfying various angle constraints and avoiding both other agents and static obstacles.

Conclusion

The goal of this work was to propose a method for collision-free multi-agent path finding in continuous environment. A Continuous-Environment Conflict-Based Search (CE-CBS) algorithm was proposed, which is based on existing CCBS [Andreychuk et al., 2022] and RRT* [Karaman and Frazzoli, 2011] algorithms while also using a smoothing mechanism. Agents in this model communicate via CE-CBS and plan conflict-free paths. Individual paths are planned using RRT*, which is modified to take CE-CBS constraints and kinematic constraints into the account. The output of this algorithm is then smoothed with B-spline curves.

Various parameters of the proposed model were studied to determine how its behavior changes based on both their setting and on the environment and agent positioning. Agents were shown to be able to navigate planned sets of smooth curves while keeping the path free from any obstacles, both static and dynamic, where dynamic obstacles are other agents present in the environment.

Future work should entail extending the model to contain a wider range of kinematic constraints, but also nonholonomic constraints, to more closely approach reality-based scenarios, where for example several autonomous nonholonomic vehicles may be present.

Bibliography

[Andreychuk et al., 2021] Andreychuk, A., Yakovlev, K., Boyarski, E., and Stern, R. (2021). Improving continuous-time conflict based search.

[Andreychuk et al., 2022] Andreychuk, A., Yakovlev, K., Surynek, P., Atzmon, D., and Stern, R. (2022). Multi-agent pathfinding with continuous time. *Artificial Intelligence*, 305:103662.

[Ayawli et al., 2019] Ayawli, B., Mei, X., Shen, M., Appiah, A., and Kyeremeh, E. F. (2019). Optimized rrt-a* path planning method for mobile robots in partially known environment. *Information Technology And Control*, 48:179–194.

[Bonabeau, 2002] Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(suppl_3):7280–7287.

[Cao et al., 2023] Cao, N., Yi, G., Zhang, S., and Qiu, L. (2023). A multi-objective path-smoothing algorithm based on node adjustment and turn-smoothing. *Measurement and Control*, 56(7-8):1187–1201.

[Cohen et al., 2021] Cohen, L., Uras, T., Kumar, T. K. S., and Koenig, S. (2021). Optimal and bounded-suboptimal multi-agent motion planning. In *Symposium on Combinatorial Search*.

[Dubins, 1957] Dubins, L. E. (1957). On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516.

[Eshtehardian and Khodaygan, 2022] Eshtehardian, A. and Khodaygan, S. (2022). A continuous rrt*-based path planning method for non-holonomic mobile robots using b-spline curves. *Journal of Ambient Intelligence and Humanized Computing*, 14.

[FIT ČVUT, 2024] FIT ČVUT (2024). CloudFIT.

[Garrido Carpio, 2018] Garrido Carpio, F. (2018). *Two-staged local trajectory planning based on optimal pre-planned curves interpolation for human-like driving in urban areas.* PhD thesis.

[Hart et al., 1968] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107.

[He et al., 2016] He, L., Pan, J., and Manocha, D. (2016). Efficient multi-agent global navigation using interpolating bridges. *CoRR*, abs/1607.07472.

[Jiang and Wu, 2019] Jiang, J. and Wu, K. (2019). Cooperative pathfinding based on memory-efficient multi-agent RRT. *CoRR*, abs/1911.03927.

[Karaman and Frazzoli, 2011] Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *CoRR*, abs/1105.1186.

[Lai et al., 2023] Lai, R., Wu, Z., Liu, X., and Zeng, N. (2023). Fusion algorithm of the improved a* algorithm and segmented bézier curves for the path planning of mobile robots. *Sustainability*, 15:2483.

[Lan and Di Cairano, 2015] Lan, X. and Di Cairano, S. (2015). Continuous curvature path planning for semi-autonomous vehicle maneuvers using rrt. In *2015 European Control Conference (ECC)*, pages 2360–2365.

[LaValle, 1998] LaValle, S. M. (1998). Rapidly-exploring random trees : a new tool for path planning. *The annual research report.*

[Li et al., 2014] Li, J., Liu, S., Zhang, B., and Zhao, X. (2014). Rrt-a* motion planning algorithm for non-holonomic mobile robot. In *2014 Proceedings of the SICE Annual Conference (SICE)*, pages 1833–1838.

[Neto et al., 2010] Neto, A. A., Macharet, D. G., and Campos, M. F. M. (2010). Feasible rrt-based path planning using seventh order bézier curves. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1445–1450.

[Piegl and Tiller, 1996] Piegl, L. and Tiller, W. (1996). *The NURBS Book.* Springer-Verlag, New York, NY, USA, second edition.

[Russell and Norvig, 2020] Russell, S. and Norvig, P. (2020). *Artificial Intelligence: A Modern Approach.* Prentice Hall, 4 edition.

[Savla et al., 2005] Savla, K., Frazzoli, E., and Bullo, F. (2005). On the point-to-point and traveling salesperson problems for dubins' vehicle. volume 2, pages 786 – 791 vol. 2.

[SciPy, 2023a] SciPy (2023a). *Scipy.interpolate.splev - SciPy v1.11.4. Manual.*

[SciPy, 2023b] SciPy (2023b). *Scipy.interpolate.splprep - SciPy v1.11.4. Manual.*

[Sederberg, 2012] Sederberg, T. W. (2012). *The NURBS Book.* Brigham Young University.

[Sharon et al., 2014] Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. (2014). Conflict-based search for optimal multi-agent pathfinding.

[Surynek, 2010] Surynek, P. (2010). An optimization variant of multi-robot path planning is intractable. In *AAAI*.

[Surynek, 2020] Surynek, P. (2020). Swarms of mobile agents: From discrete to continuous movements in multi-agent path finding. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3006–3012.

[Čáp et al., 2013] Čáp, M., Novák, P., Vokřínek, J., and Pěchouček, M. (2013). Multi-agent rrt*: Sampling-based cooperative pathfinding (extended abstract). *CoRR*, abs/1302.2828.

[Čáp et al., 2015] Čáp, M., Vokřínek, J., and Kleiner, A. (2015). Complete decentralized method for on-line multi-robot trajectory planning in valid infrastructures. *CoRR*, abs/1501.07704.

[Verbari, 2017] Verbari, P. (2017). *Multi-RRT* : a sample-based algorithm for multi-agent planning*. PhD thesis.

[Váňa and Faigl, 2015] Váňa, P. and Faigl, J. (2015). On the dubins traveling salesman problem with neighborhoods. pages 4029–4034.

[Walker et al., 2018] Walker, T. T., Sturtevant, N. R., and Felner, A. (2018). Extended increasing cost tree search for non-unit cost domains. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 534–540. International Joint Conferences on Artificial Intelligence Organization.

Appendix **A**

List of used shortcuts

CBS	Conflict-Based Search
CCBS	Continuous Conflict-Based Search
CE-CBS	Continuous-Environment Conflict-Based Search
MAPF	Multi-Agent Path Finding
RRT	Rapidly Exploring Random Tree

Appendix **B**

Contents of attachments

```
readme.txt ........................ the file with attachments description
src ....................................... the directory of source codes
├── impl .................................. the directory of the program
└── thesis ............... the directory of LaTeX source codes of the thesis
    ├── img .................................. the thesis figures directory
    └── *.tex .................... the LaTeX source code files of the thesis
text ........................................ the thesis text directory
└── thesis.pdf ...................... the Diploma thesis in PDF format
```