"RRT-FT & RRT-TP;
New Approaches to Time-based Motion Planning"

M. de Rie
Game and Media Technology
Utrecht University

October 2012

Introduction

This thesis concludes the work that I have done between January 2012 and September 2012 at the National Aerospace Laboratory of the Netherlands (Nederlands Lucht en Ruimtevaart Laboratorium, NLR) as my master thesis, which is made up out of two parts; the work that I have done within the context of the C-Share project and the research that was done in succession to this.

The C-Share project concerns itself with developing automated tools for Air Traffic Management that plans paths in 2D+time while keeping Air Traffic Controllers involved in the decision making. My part in this project was to implement a module which is capable of planning paths for aircraft, which are then presented to the Air Traffic Controllers. The module that I have implemented contains a modified version of the Rapidly-exploring Random Tree algorithm [9], which performs well in the C-Share context, but also exposed the need for better path planning algorithms, which leads to the second part of this thesis.

In my research I have developed two algorithms which are capable of planning paths in the 2D+time amongst moving obstacles, which can be travelled at a constant speed which is chosen from an interval of speeds which is provided as a part of the solution. This type of planning originates from the C-Share context and can be used in Air Traffic Managements, since it not only allows an entity to travel at a constant speed, it also lets the entity pick an arrival time. This is particularly useful when aircraft need to get back, or stay on schedule, so that delays can be avoided and future conflicts between aircraft and a delayed aircraft can be eliminated. The algorithms that were developed were tested on a number of different sets of scenarios and the performance was measured by recording the rate of success, the time it took to find a solution, and the length of the shortest path.

Contents

1 The C-Share Project

This thesis is made up out of two parts; the work that was done for the C-Share project and the research that was done in succession to this. In this section I will go into a high level description of the C-Share project and its context, such as the field of air traffic management and the consortium partners. Next, I will introduce the concept of the Joint Cognitive System, followed by a description of my involvement in the project.

1.1 Air Traffic Management

Air Traffic Management (ATM) is the regulation of civilian air traffic done by Air Traffic Controllers (ATCos), who have the responsibility of making sure that all air traffic is guided in such a way that no dangerous situations arise. In order to do this, ATCos are in contact with the aircraft and can instruct a pilot to make changes to, for instance, his aircraft's speed or direction. ATM is considered to be a difficult and responsible task that is becoming increasingly more complicated due to the constant increase of the amount of air traffic.

Taking the growth of the air traffic sector into consideration, it is important to keep looking for solutions that can aid ATCos with finding solutions for potentially dangerous situations [4]. The C-Share project concerns itself with designing a shared representation of the air space, in order to present comprehensible resolutions for conflicts to the ATCos. This project is executed by a consortium between the NLR , Thales Netherlands, and Delft University of Technology. The project has started in May of 2011 and has a duration of 30 months.

1.2 Joint Cognitive System

The top priority of the C-Share project is to develop a piece of automation that can assist ATCos, which is designed according to the principle of the Joint Cognitive System (JCS). The idea of a JCS is that both the automated agents and the users use that same representation of the world in order to avoid discrepancies between their views on the situation. Secondly, the automated agents are not allowed to implement any solutions; the solutions it produces are presented to the ATCos who must then make the final decision, opposed to *free-flight* models as seen in [12]. The way that the solutions of the automated agents are presented to the users, is targeted to be so that the user is shown the reasoning behind it, thus allowing for a deeper insight into the decision making of the automated agents.

The JCS is made up out three elements; the environment, the automated agents, and the human actors. The environment stores all the data on the world and is fed by different servers, such as a traffic server and a weather server, and shares its information through the Functional Airspace Representation (FAR) and the Functional Airspace View (FAV), with the other elements.

1.3 Involvement in the Project

An essential part of the C-Share project is the planning of trajectories for aircraft at a timescale of roughly twenty minutes. This involves both hard and soft constraints that should be taken into account. Each suggested trajectory must be flyable and respect all separation criteria with regard to all other aircraft without exception. There are normally two separation criteria; aircraft may not approach each other closer than a distance ls in the earth's plane, also called lateral separation. The second criterion is that aircraft must also maintain a vertical separation of a certain distance vs. Next to these hard constraints, there are soft constraints such as the interests of multiple parties which must be looked after, as for instance the air traffic controller, the airline, and the pilots. Each party brings

new constraints and/or preferences to the table which will possibly conflict with the ones from other parties. An example of this is that an airline will prefer to minimize fuel burn, while an ATCo will give preference to optimizing the comprehensibility which may not be compatible with the airlines wishes. The objective of the C-Share project was to design and implement the first version of the automated agents that is part of the JCS, which is responsible for creating the before mentioned trajectories.

2 The C-Share Problem

In the previous section the C-Share project was explained and we will continue on this in the next section. I will put forward the software that was implemented within the scope of the C-Share project, followed up with the difficulties that were encountered during the work on this part of my internship. This will lead to the motivation for the need to develop new algorithms, succeeded by the problem formulation and eventually the research question.

2.1 Problem Description

For the first version of the JCS, it was decided to simplify the original problem in order to put more emphasis on the collaboration between the different parts of the system. For this version of the JCS, the problem was defined as follows; given a set of aircraft travelling in a straight line at a constant speed, find all conflicting aircraft and be able to suggest a new trajectory for one of the conflicting aircraft. We chose to let all aircraft travel in the same plane and each aircraft was represented by a disc centered at the aircraft position with a radius $0.5 * ls$, where ls is the minimal required lateral separation. Each solution was required to, at some point, resume the trajectory it was replacing in both space and time. Secondly, each solution can only use two speeds, the original speed at which at it was travelling and another one to resolve the conflict. Since the original trajectory must be resumed, each solution must be made up out of three parts; a part of the original trajectory with the original speed, a trajectory with a different speed that resolves the conflict, and another trajectory from the original path at the original speed. Next to these constraints, the new trajectory may only implement speeds which are within the performance model of the aircraft for which that trajectory is intended. Lastly, the module is required to be able to calculate the cost of the new trajectories with an adaptable cost function, and be able to selected the trajectories that have the lowest cost. In this implementation a simple cost function was used, namely the length of the trajectory, because developing more complicated cost functions is outside the scope of this thesis.

2.2 Literature

The problem described in the previous section is a path planning problem in two dimensions plus a time dimension. In this section I will go into the different classes of path planning algorithms that can be used in order to solve our problem, and discuss the suitability of each class by considering their pros and cons.

2.2.1 Potential Field Methods

Potential Field Methods is a class of path planning algorithms that relies on attractive and repellant forces to guide an entity through the configuration space. For instance, the goal state would have an attractive force, opposed to obstacles who exert a repellant force. The potential field created by the forces on the configuration space can be searched using algorithms such Hill climbing or Gradient descent. These methods are known to be vulnerable to local minima which is a great disadvantage, since it would mean that the goal configuration is not reached. More importantly, combining the time-based aspect of our problem with a potential method, is not trivial since a force does not take into account that an obstacles is moving. In other words, a path of a moving obstacle can not be anticipated through a force, meaning that a potential field method might not be possible or requires a lot of adaptations. Despite the fact that these disadvantages make it very unattractive to develop a potential field algorithm for our problem, there are some advantages to it as is shown in [10]. An example of this is that these methods produce smooth, curved paths by themselves, which

might be required in a later stage. However, the pros do not outweigh the cons thus this class of path planning techniques was not chosen.

2.2.2 Roadmaps

Roadmaps by itself is not a path planning technique, but merely the concept of building a graph in the configuration space that describes how various configurations are connected to each other. If all configurations of a roadmap are connected, a path can be found if the start and goal configuration can be connected to the roadmap. A roadmap can typically be used for multiple queries if required.

2.2.3 Cell Decomposition Methods

Cell Decomposition Methods are based on the principle of dividing the configuration space into simple cells which are then connected to each other forming a roadmap. An important feature of this class of methods is that they guarantee completeness, which means that a solution will always be found if one exists. Despite this great advantage, CDM do have a considerable number of disadvantages when applying them to our problem. For one, CDM will become progressively harder to implement when incrementing the number of dimensions of the problem, which might provide problems in the future when expanding the problem. Secondly, CDM can require a lot a time to run since the whole configuration space is indexed, which may not be necessary. Overall, I did not choose to develop a cell decomposition method due to the first stated disadvantage, because I believe it is important that the developed method is scalable.

2.2.4 Grid-Based Methods

Grid-based Methods (GBM) or Grid-based search methods place a grid over the configuration space and build a graph between adjacent points of the grid in order to build a graph for which a solution can be extracted. When a coarser grid is used, the algorithm will become faster at the expense of the possibility to find a path through more narrow passages. A great downside of this kind of algorithm is that the number of points in the grid grows exponentially with the number of dimensions, which makes it unsuitable for future expansions. In the C-Share project we are dealing with three dimensions (2D+time) but this could be scaled to include more complex flight dynamics such as the pitch, roll, and jaw of a an aircraft, up to the angular acceleration which might considerably raise the number of dimensions. The fact that GBM are easy to implement and guarantee completeness within the resolution of the grid, does not outweigh the downside, thus this kind of method was not chosen.

2.2.5 Sampling-Based Methods

Sampling-based Methods can be roughly classified into two categories; Probabilistic Roadmaps (PRM) and Rapidly-exploring Random Trees (RRT). Both these algorithms rely on a graph or tree that is expanded by attempting to add and connect randomly chosen configurations to it. A direct downside of this is that sampling-based methods do not guarantee completeness. However, sampling-based methods can often proven to be probabilistically complete, which means that the probability that a solution will be found, given that at least one exists, will converge to 1 as the number of added configurations will approach infinity. One advantage of sampling-based methods is the fact that they can easily be adapted to higher dimensions, because these methods simply deal with configuration-configuration connections which makes the amount of work proportional to the number of configurations that is added. This the opposite of for instance cell decomposition methods or grid-based methods

were the amount grows exponentially with the number of dimensions. The configuration-configuration connections are left up to the local planner, which is a method that determines whether a connection can be travelled within the constraints of the problem, such as obstacles or speed constraints. Local planners often simply use a linear interpolation between two configurations, but can be expanded to deal with any kind of interpolation which is useful when dealing with our problem, since it makes it possible to incorporate flight dynamics as complicated as desired. Having a local planner incorporated into the algorithm is an advantage, since it means that the algorithm need not be adapted when the dimensionality of the problem is changed, because the local planner can simply be replaced for one that does handle the new dimensionality. Overall the sampling-based methods score very positively compared to the other class of path planning algorithms. On top of this, previous work done at the NLR [14] shows that both PRM and RRT are suitable for solving our problem. In the following to paragraphs I will go into the differences between PRM and RRT.

Probabilistic Roadmaps This technique builds a roadmap in the configuration space by attempting to add randomly chosen nodes to a graph and later connecting the start and goal configuration to the graph. The main difference with RRT, is that PRM directly connects to the new candidate node whereas RRT use a less greedy approach. Another difference is that PRM typically connect multiple times to a newly added node. PRM are a multiple-shot approach which means that a roadmap can be used multiple time since it is not dedicated to a specific pair of start and goal configurations, opposed to RRT which is a single-shot approach.

Rapidly-exploring Random Trees Rapidly-exploring Random Trees or Rapid Random Trees (algorithms 1,2) is a form of probabilistic path planning which has been first described by laValle [7]. The idea of this method is to explore the configuration space in a non-greedy fashion, in order to make a graph that connects the start and goal configuration. This is done by starting with the start configuration and repeatedly placing a randomly chosen configuration within a distance u of the closest configuration in the graph, followed by attempting to connect the two, as seen in algorithm 1. As the algorithm progresses throughout the configuration space it will reach the goal with probabilistic completeness. When this happens, the adding of nodes is stopped and the graph can be traversed for the solution. A note must be made about the metrics that are used for determining which configuration in the graph is the closest to the random configuration, and for place the random configuration at a distance u. Namely, this is not trivial when using dimensions of different nature such as euclidean dimensions versus time dimensions. In this thesis all distances between configurations will only be measured in the euclidean dimensions.

The largest difference with PRM is that RRT use non-greedy exploration, which trades speed for a higher probability of adding a new node. This property introduces a new concept named Voronoi Bias, which is another way of saying that the probability of exploring towards an unexplored part of space, is proportional to the size of that chunk of space. This can be visualized by imagining a graph and dividing all of the space according to a Voronoi diagram using the nodes of that graph as its seeds. A larger cell in this diagram can be said to indicate an unexplored piece of space, since each cell holds exactly one node and thus larger cells are less explored. Since the probability of picking a random configuration from a cell is equal to the size of that cell divided by the size of the total space, less explored cells have a higher probability of containing the next random configuration and thus have a higher probability of being explored. This effect is called a Voronoi bias.

Another consequence of using non-greedy exploration is that the graph of an RRT typically contains shorter edges than those in a roadmap made with PRM. In the context of ATM this can be considered to be a downside of RRT, since ATCos prefer trajectories that

consist of longer, and thus fewer edges because this means less workload and a more comprehensible situation. However, making shortcuts on a solution will reduce the number of edges in a solution thus making it compatible with the ATCos preferences.

A third advantage of RRT over PRM is that the fact that RRT attempt to connect the start and goal configuration, which can be exploited by introducing a bias towards the goal in the sampling of the new randomly chosen nodes, or even grow two graphs towards each other [6]. This is not as easy with PRM since a roadmap, by definition, is a multi-shot technique which therefore can not be biased towards a goal since a multi-shot technique does not take a goal configuration into account. Having a goal bias is a heuristic which reduces the time needed for a solution to be found [13].

Both RRT and PRM have shown able to cope with higher dimensionality, however RRT have the advantage of being a single-shot technique, which means that the graph can be directed. This is already the case with most RRT because their graph is a tree, but this especially important when applying it to kynodynamic, or otherwise time-involved path planning [1], [8], [2], even though PRM have been applied to these kinds of problem as well [3].

I chose to make an implementation of a RRT instead of a PRM because of the fact that the C-Share problem requires only a single-shot technique, since each query for a new trajectory often contains a different set of obstacles and a different start and goal configuration. The C-Share project also require the method to involve a time dimension which results in the fact that configuration-configuration connections become directed. This is in favor of RRT because it uses a directed graph by nature, namely a tree.

Algorithm 1 BUILD_RRT(x_{init})

```
τ.init(x_init)
for k = 1 to K do
    x_rand ← RANDOM_STATE()
    EXTEND(τ,x_rand)
return τ
```

Algorithm 2 EXTEND(Graph τ , Node x_{rand})

```
x_near ← FINDNEARESTNODE(x_rand)
if DISTANCE(x_near, x_rand) > u then
    Move x_rand towards x_near until DISTANCE(x_near, x_rand) = u
if EDGE(x_near, x_rand) is does not intersect any obstacles then
    ADDCONFIGURATION(x_rand)
    ADDEDGE(x_near, x_rand)
    if x_rand has not been moved then
        return Reached                          ▷ x_rand has been reached.
    else
        return Advanced          ▷ The graph advanced in the direction of x_rand.
else
    return Trapped                              ▷ No progression has been made
```

2.3 Module Description

In the previous section I have concluded that RRTs are the most suitable for solving the problem stated in section 2.1. In this section I will elaborate on the implementation of an

adaptation of the RRT algorithm for this problem. However, I will first go into the part of the module that takes care of the communication with the rest of the JCS.

2.3.1 TCP/XML Communication

The module communicates with the rest of the system using a TCP connection, which was chosen so that the different modules of the JCS could be written in any programming language on any platform. Another advantage is that a TCP connection can be established over the internet or LAN, through a cable, or within one computer, which facilitates the debugging and testing sessions.

It was agreed upon that the format of the data that is sent through the TCP connection is in the XML-format. The developed module is a passive module in the sense that it only responds to calls instead of spontaneously generating calls. The only exception to this is during the initialization of the module, when it asks the server for the data of the airspace and data concerning the coordinate system.

Call description Every XML-file which is transferred between the modules of the JCS is required to contain the following (meta-)data;

Call ID This is used to link an answer to a question.

Function Name When the module is initialized it will request two things from the JCS, a description of the boundaries of the airspace and the center of projection used for the geographic coordinate system.

2.3.2 Path Planning

The path planning function of the module can be called in order to resolve a conflict by the JCS. The call takes a callsign of an aircraft in order to identify it, ac_c, and the current time, t_c, as arguments. The conflict detection function is called in order to check whether ac_c has any upcoming conflicts within a certain time from t_c which can be taken into account. The idea of the function is to remove the part of the original trajectory which contains the conflict, and fill the resulting gap with a trajectory that can be flown at a constant speed using a bidirectional RRT [5] (algorithm 4).

Bidirectional RRT Since it is preferable to resume the original path after avoiding a conflict, we chose to use the bidirectional variation of the RRT algorithm (algorithm 4). This variation can start with either two nodes or two graphs, which are grown alternatingly towards each other until they become connected. A trajectory can be though of as a graph, so when a piece of a trajectory is removed from the middle, the result are two graphs. This is done in the context of the C-Share project, were the two starting graphs are obtained by taking the original trajectory and removing the part which contains a conflict. These two pieces of trajectory are then connected again using a biRRT. This is advantageous over the original RRT because the solution produced by a biRRT does not necessarily contain every node from the starting graphs, such that the number of possible solutions becomes larger opposed to when an original RRT would have been used to connect x_f to x_l. In other words, the biRRT is given freedom in terms of how much of the original trajectory is used.

Path Recycling The RRT is only able to connect two nodes together instead of connecting a node halfway to an edge. In order to overcome this, extra nodes were interpolated on the original path of ac_c, $ac_c.p_0$ (algorithm 3 line 10). This gives the biRRT more options on

Algorithm 3 PLANPATH(Aircraft ac, t_{offset}, Traffic T)

define $ac.p_0 : \{x_0, \ldots, x_n\}$ as the original trajectory of the aircraft
if ac is involved in a conflict **then** ▷ This is checked in a manner comparable to algorithm 5
    $t_f \leftarrow$ find the first moment that ac is involved in a conflict
    $t_l \leftarrow$ find the last moment that ac is involved in a conflict
    $t_f \leftarrow \max(t_f - t_{offset}, x_0.time)$
    $t_l \leftarrow \min(t_l + t_{offset}, x_n.time)$
    $x_f \leftarrow$ interpolate t_f in $ac.p_0$
    $x_l \leftarrow$ interpolate t_l in $ac.p_0$
    insert x_f and x_l in $ac.p_0$
    interpolate nodes every 30 seconds along $ac.p_0$
    remove all nodes between x_f and x_l
    define τ_a and τ_b as a 2D+T graph
    $\tau_a \leftarrow \{x_0, \ldots, x_f\}$
    $\tau_b \leftarrow \{x_l, \ldots, x_n\}$
    define p_{new} to be a trajectory in 2D+T ▷ Not every configuration has to have a time.
    $p_{new} \leftarrow$ RUNBIDIRECTIONRRT(τ_a,τ_b) ▷ In 2D
    $x_i \leftarrow$ the first node from p_{new} which does not have a time
    $x_j \leftarrow$ the last node from p_{new} which does not have a time
    $gap.length \leftarrow$ the length of $\{x_{i-1}, \ldots, x_{j+1}\}$
    $gap.time \leftarrow x_{j+1}.time - x_{i-1}.time$
    $gap.speed \leftarrow gap.length/gap.time$
    for $s \leftarrow i$ **to** $j+1$ **do**
        $x_s.time \leftarrow x_{s-1}.time +$ DISTANCE$(x_{s-1}, x_s)/gap.speed$ ▷ Adding times to the new configurations
    if $p_{new} \cap (T - ac) = \emptyset \wedge p_{new}$ is flyable according to BADA **then** ▷ Validation
        return p_{new}

Algorithm 4 RUNBIDIRECTIONALRRT(Graph τ_a, Graph τ_b, Node x_{start}, Node x_{goal})

for $k = 1$ **to** K **do**
    $x_{rand} \leftarrow$ RANDOM_STATE()
    if $\neg$EXTEND$(\tau_a, x_{rand}) = Trapped$ **then**
        if EXTEND$(\tau_b, x_{rand}) = Reached$ **then**
            return PATH(x_{start},x_{goal})
    SWAP(τ_a,τ_b)
return $Failure$

how much of the original path is reused. The second argument of planPath, t_{offset}, can be used to control how much of the original path is being used for the biRRT method.

Flight Dynamics At this time, the module only verifies whether the aircraft is capable of flying the different speeds used in a solution. This is done by a piece of software which is a variation of the the Eurocontrol Base of Aircraft Data (BADA).

Parameters The PLANPATH function has two variables which are not provided by the call, t_{offset} and, the maximal edge length u. For each variable a for loop was implemented that iterates over a range of values. The boundaries of this range and the step size are defined in an external file, so that they can easily be modified if needed.

2.3.3 Conflict Detection

When the module receives a request to detect conflicts, it will request the current state of the traffic from the traffic server. After the traffic has been received, all conflicts occurring within the look-ahead time are detected using algorithm 5. The look-ahead time is an optional argument of the method and is by default set to 20 minutes.

Algorithm 5 DETECT_CONFLICTS(Traffic T, look-ahead time t_{la}, the current time t_c)

```
1: for all Aircraft ac_1 ∈ T do
2:     for all Aircraft ac_2 ∈ T do
3:         if ac_1 and ac_2 intersect at a time within t_la from t_c then
4:             report conflict(ac_1,ac_2)
```

Each conflict that is reported contains following information;

- the callsigns of the involved aircraft
- the begin time
- the end time
- the smallest distance between the involved aircraft

2.4 Conclusions

The adaptation of the biRRT that was implemented together with the function for detecting conflicts showed to be sufficient during the first trials of the JCS. Furthermore, the literature showed that there do not yet exist methods able of planning paths that can be travelled at a constant speed amongst moving obstacles. Though the implementation of the module yielded valid solutions, it does still suffer from several disadvantages such as not being able to make shortcuts without introducing an extra speed that should be flown, or the fact that the method will occasionally reject a candidate solution.

3 Problem Formulation

From the literature study that was done in the C-Share project, it can be concluded that there does not yet exist a solution targeted for motion planning with a constant, yet unknown speed. This proved to be a difficulty when doing 2D+T motion planning, because in order to get a constant speed, the speed profiling had to be done afterwards, which meant that the edges in the graph had to be validated after a candidate solution had been built. This was unsatisfactory since it meant that every so often, candidate solutions would be discarded.

This is why I have committed the research I have done during my time at the NLR, to developing algorithms that would be able to perform this kind of motion planning. Having these kinds of planning algorithms will be valuable for applications such as ATM due to constraints in this field.

Since the C-Share project deals with a large number of both soft and hard constraints, I will put forward the exact requirements for the new algorithms which will contain not all constraints and some will be simplified. After this I will discuss the goals of the experiments and the measurements that will be done during the experiments.

3.1 Problem Description

3.1.1 Configuration Space

In the C-Share project, both the moving obstacles and the entity for which new paths are planned live in a 2D+time dimensional space, which we will call C_{space}. The original problem of planning paths for actual aircraft is in 3D+time, but this was simplified. The algorithms that will be developed will be required to be able to plan paths in 2D+time, similar to the C-Share project, although the dimensionality should be scalable. Therefore the configuration space of this problem and thus for the experiments is a 2D+time dimensional space. A part of C_{space} is defined as C_{room} which has boundaries in the two euclidean dimensions; $[0, x_{max}]$ and $[0, y_{max}]$, and $[0, \rightarrow\rangle$ for the time dimension.

3.1.2 Obstacles

The motion planning problem in the C-Share project sets itself apart from most other motion planning problems because the obstacles are moving and each one has a fully known trajectory in 2D+time. The set of moving obstacles with a known trajectory is called O, and lies fully within C_{space} but not per se fully in C_{room};

$$O \subset C_{space}$$

and

$$O \cap C_{room} \neq \emptyset$$

Other kinds of obstacles such as static obstacles or moving obstacles with an unknown or partially known path are outside of the scope of this thesis.

The moving obstacles that are considered in this thesis have the same shape as the obstacles in the C-Share project, namely a disc shape. This comes from the lateral separation criterion that should always be maintained between aircraft. The radius of the disc is equal to the lateral separation distance whereas normally this would be half of the lateral separation distance. This is explained further on when the entity for which the paths are planned is introduced. Regards of this, the algorithms are required to be designed such that they are adaptable to any kind of shape.

To summarize the properties of the obstacles,

- moving with a known trajectory

- the trajectory is in 2D+time
- linear interpolation between points of the trajectory
- represented by a disc with radius equal to the lateral separation distance

3.1.3 Entity

The entity for which the path is planned is an abstraction of an aircraft simlilar to the moving obstacles, and is called e. In order to achieve a formal definition, I will state the properties of e from scratch, instead of simplifying the C-Share context.

First off, the shape of e was originally a disc with a radius of half of the lateral separation distance, but this is reduced to a point. This is done by taking the Minkowsky sum of the original shape of an obstacle and the entity which both are a discs with a radius of half the lateral separation distance. The result of the Minkowsky sum is that the obstacles becomes discs with the lateral separation distance as the radius, and the entity becomes a point. From this, a requirement for a valid path is that it should not overlap with any obstacle, or in other words the entity may only travel through the free configuration space, C_{free}, which is defined as follows.

$$C_{space} \setminus O = C_{free}$$

which leads to

$$e \cap O = \emptyset$$

and

$$e \subset C_{free}$$

Secondly, the entity has speed limitations similar to those of an aircraft, namely a minimum and maximum speed called v_{min} and v_{max}. This means that at no point the entity will be able to travel faster than v_{max} or slower than v_{min}.

Next to these constraints, e has a few other properties which can be seen as simplifications of the original properties of an aircraft. Namely, e has instantaneous acceleration which means that it is able to change the direction in which it is heading instantaneously and the same goes for its speed, even though speed changes will not be used.

Finally, e is only allowed to navigate through a confined part of C_{space} which is called the room, or C_{room}.

$$e \subset C_{room} \subset C_{space}$$

To summarize the properties of the entity,

- point shape
- may not intersect with O
- instantaneous accelaration
- can not travel faster than v_{max} or slower than v_{min}
- can only travel within C_{room}

3.1.4 Solutions

In the C-Share project some of the preferences of ATCos and other involved parties in that project are discussed. From this can be learned that the problem that is targeted within that project has to deal with a large number of both requirements and preferences. Here I will put forward the requirements for the solutions that the algorithms should produce, which are for the most part based on the requirements and preferences from the C-Share project.

Take note that a number of requirements and constraints have already been discussed in this section. I will not discuss all of these requirements and constrains again, though I will repeat some of them.

The most important constraint that all solutions should implement, is that all proposed paths should be able to be travelled at a constant speed. This originates from the preferences of ATCos who prefer to adjust one thing at a time such as the speed, direction or flight level (height) of an aircraft. On top of this, the solutions will be required to be able to be travelled at a constant speed picked from an interval of speeds that should be provided as a part of each solution. This means that every solution is made up out of two parts; a path in 2D called P_{2D} and an interval of speeds called V, such that P_{2D} can be travelled at a constant speed v_i with $v_i \in V$, without violating any constraints stated in this section.

Having an interval V from which a speed can be chosen, yields a number of implications when applying this kind of solution to the context of ATM. First off, this gives an ATCo a range of solutions from which he can chose, allowing him to adapt the solution to his preferences. Secondly, having an interval of speeds to chose from, also implies an interval of times at which e can arrive at x_{goal}, which can be of useful when an aircraft is required to arrive at a certain place at a certain time.

To summarize the constraints concerning the solutions,

- Provide a path P_{2D} and an interval of speeds V
- P_{2D} can be travelled at a constant speed $v_i \in V$

Next to all of the above stated requirement, shorter solutions are preferred for a number of reasons. Firstly, shorter solutions are often preferred by airlines due to the reduction of fuel burn, which is also a plus for the environment. On top of this, measuring the length of the trajectories is an comprehensive way of comparing solutions with each other. This is why it was chosen to measure the length of the shortest path of each graph that is built by an algorithm, as the way to measure the quality of the solution.

- Measure the shortest path as an indicator of the quality of the solution.

There are of course many other ways of measuring the quality of a solution, which can also be relevant to the context of ATM as well. For instance, measuring the number of nodes in P_{2D} or the size of V will both provide valuable information about the solution, namely the number of nodes is proportional to the number of instructions that an ATCo will have to give to a pilot in order to implement that solution. Secondly, the size of V not only indicates the range of speeds at which P_{2D} can be travelled, it also gives an indication of how much space an aircraft has to maneuver when travelling along P_{2D}. Both of these ways of measuring the quality of the path will be further discussed in section 8. The reason why it was chosen to use one metric for measuring the quality of the solution is because when using multiple metrics it is no longer trivial to determine the best solution. Using multiple metrics will yield a best solution for each metric, and this will often not be the same solution. This leads to the fact that when presenting the results of the quality measurements, it is unclear how well the best solution using metric A scores when using metric B. It is possible to overcome the problem of having multiple best solutions by combining multiple metrics into one, but this is outside the scope of the thesis.

3.1.5 Shortcuts

The last requirement for the new algorithms is that they allow for shortcuts to be made on a solution. Due to the probabilistic nature of RRT, the shortest path often is not found. However, by making shortcuts on a solution, the length of the shortest solution can often be reduced. There are more reasons to make shortcuts such as the fact that they allow a

solution to reduce the number of nodes in it, which might be preferred depending on the application.

- The new algorithms are required to allow shortcuts to be made

3.2 Research Question

The literature study that was done within the C-Share project points out that it can be concluded that there not yet exists a solution for planning paths which can be travelled at a constant speed. Existing techniques from the field of kynodynamic path planning only allow for concrete actions instead of keeping things in middle such as speed. From studying the different kinds of approaches to the path planning problem (section 2.2), I concluded that a sampling-based technique is the most suitable for this kind of problem, due the fact that this class handles higher dimensions really well, which makes for expandability in the future. However, sampling-based algorithms which can be used for this specific kind of problem do not yet exist. More importantly, from the sampling-based path planning algorithms, RRT showed to be the most suitable for our problem (see section 2.2.5). This is why I pose as the main research question; "Can the RRT algorithm be adapted so that it is able to produce solution consisting out of a path, P_{2D}, and an interval of speeds, V, such that P_{2D} can be travelled with a constant speed $v_i \in V$, while avoiding moving obstacles with a known trajectory?" For the full description of the problem requirements see section 3.1.

Next to validating the capability of the new algorithms to find solutions, the quality of these solutions will be compared by measuring the lengths of the shortest solutions. The second research question is; "Which algorithm produces shorter paths on average?"

One of the requirements for the new algorithms is that they should allow for shortcuts to be made on an existing solution. Making these shortcuts should reduce the length of the shortest path, thus the third research question becomes; "How does making shortcuts influence the length of the shortest path?"

Lastly, it is important to measure how much time the new algorithms take to make a solution, which is especially relevant for an online application such as ATM.

"How does the averaged time it takes to make a solution compare between the new algorithms?"

4 Algorithm Description

In this last section I have discussed the goals of my research; study whether RRTs can be used to produce paths that can be travelled at a constant speed. This will be the center of my research and in this section I will put forward two algorithms which were designed to solve the problem in section 3. Both algorithms will be discussed in detail with the aid of figures and pseudo code. After this section I will present the experimental setup followed by my findings.

4.1 Rapid Random Tree: Forbidden Times

In this subsection I will put forward the first of the two algorithms that I have developed; the Forbidden Time variation on the RRT algorithm; RRT-FT. In this method, each edge in the graph holds with it information on when e may not travel over this edge.

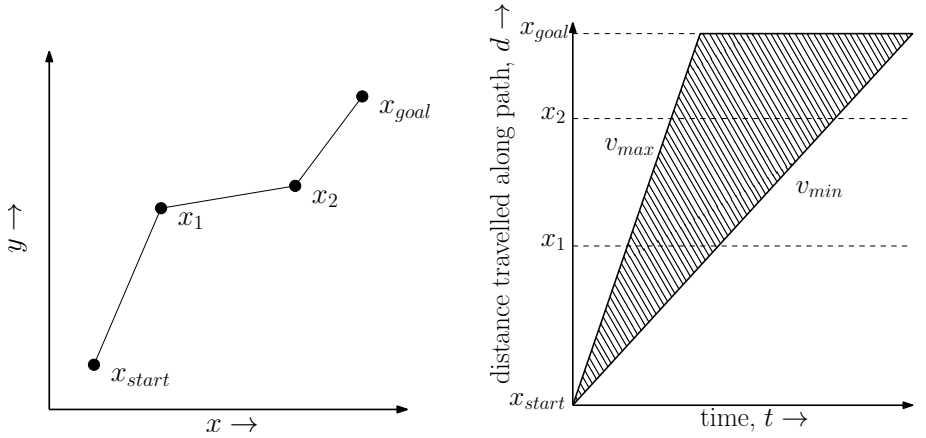
4.1.1 Idea

The principle of the RRT-FT method is to build a roadmap in the Euclidean dimensions, similar to the original RRT algorithm, with one adaptation. Each edge ed holds a collection of intervals which indicate the times at which an obstacle obstructs any part of ed. An interval of time associated to an object obstructing an edge, is called a Forbidden Time-object. The idea behind this is that any Forbidden Time-object is independent of the time at which e visits its related edge, thus giving it the advantage of being able to make shortcuts without having to update any other parts of the graph or Forbidden Time-objects. However, this also brings several disadvantages with it, such as the fact that it is not guaranteed that every node in the graph of the RRT can be visited by e while travelling at a constant speed. This is because the algorithm only stores data about the obstacles, instead of taking into account whether e can reach the next node. In order to determine whether a node is reachable from the root, the graph has to be queried. The RRT-FT algorithm is designed to be a simple, or in other words naïve, method that opposes the RRT-TP algorithm which I will present further on in this thesis.

Figure 1(a) shows a representation of a path planned in the x, y-plane in an empty environment. Given that the entity travelling this path can travel at any speed in the interval $[v_{min}, v_{max}]$, a d, t-diagram can be made (figure 1(b)), which shows the relation between the distance travelled along this specific path d, and the time t. The horizontal lines indicate a place along the path, namely the nodes. The parts of these horizontal lines that overlap with the striped triangle, indicate the times at which a nodes can be reached. This kind of diagram can be used to visualize the forbidden time-intervals (figure 2(b)), which have the shape of a rectangle in the d, t-diagram. The rectangular shape is the result of prohibiting the entity from travelling on an edge during an interval of time. In a d, t-diagram, travelling at a constant speed is shown straight line, and since e starts to move at $t = 0$, this line originates in the origin thus it is actually a half-line. In figure 2(b), the striped triangle indicates the collections of speeds that e can travel without colliding with the Forbidden Time-object.

4.1.2 Implementation Details

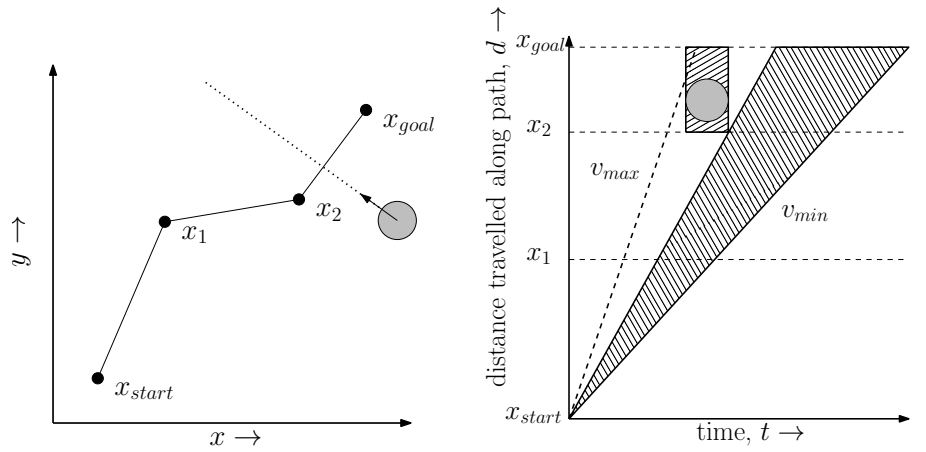
In order to implement the Forbidden Time method, I had to redefine the original extend-function, which can be seen in algorithm 2, to a new one (algorithm 6). The difference with the original version is that I have added a set of forbidden time-objects which is initialized by a function called FIND_FORBIDDEN_TIMES, which can be found at algorithm 7. Also note that opposed to the original extend function, this one does not check whether an edge intersects with any obstacles because this data is stored as Forbidden Time objects. As said



(a) A graphic representation of a path planned in an environment without obstacles.

(b) The tilled triangle shows the relation between the distance travelled along a path and the time.

Figure 1: The relation between the Euclidean space and d, t-space



(a) The same situation as seen in 1(a) with the addition of an moving obstacle (grey disc).

(b) A d, t-diagram showing a forbidden time-interval (rectangle), an obstacle (circle), and d, t-space that e can travel through without intersecting the Forbidden Time-object (triangle). The shape of the obstacle is arbitrarily chosen.

Figure 2: Two figures illustration the relation between the euclidean space and d, t-space with the addition of a moving obstacle.

before, by doing this, it can no longer be guaranteed that each node in the graph of the RRT-FT can be reached from the root when meeting all criteria from section 3.1.

Algorithm 6 EXTEND_FT(Graph τ, Node x_{rand})

```
1: x_near ← NEAREST_NODE(x_rand, τ)
2: if DISTANCE(x_near, x_rand) > u then
3:     Move x_rand towards x_near until DISTANCE(x_near, x_rand) = u
4: τ.ADD_VERTEX(x_rand)
5: fts_{near,rand} ← FIND_FORBIDDEN_TIMES(x_near, x_rand)
6: τ.ADD_EDGE(x_near, x_rand, fts_{near→rand})
7: if x_rand has not been moved then
8:     return Reached
9: else
10:    return Advanced
11: return Trapped
```

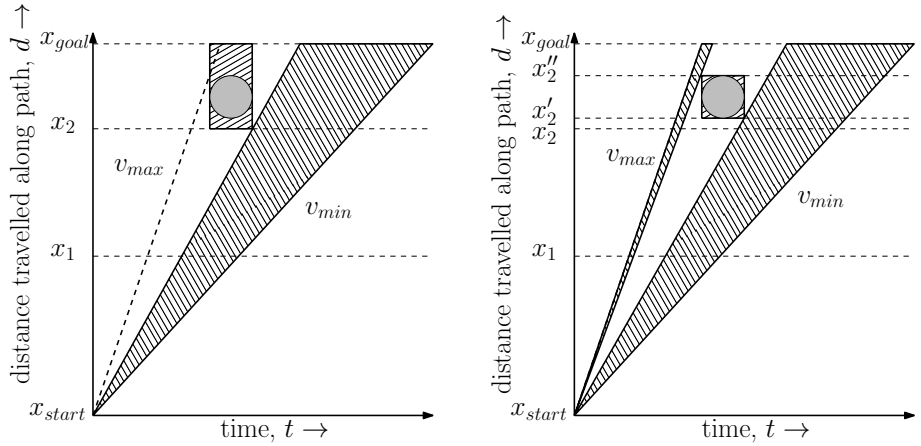
Algorithm 7 FIND_FORBIDDEN_TIMES(Node x_{from}, Node x_{to})

```
1: define an empty set of time intervals fts
2: for each obstacle o in O do
3:     for each trajectory segment ts from o do
4:         if ts crosses the path between x_from and x_to then
5:             find the begin- and end time, t_begin and t_end of the intersection
6:             fts.ADD(INTERVAL(t_begin, t_end))
7: return fts
```

4.1.3 Reducing the Size of Forbidden Times by Edge Segmentation

Due to the nature of the RRT-FT method, the forbidden space interval will always invalidate more configuration space than necessary. This is an inherent problem since the forbidden time interval is a rectangular object and the obstacles are circular objects[1]. The alternative to this rectangular shape would be to use the tangents to the obstacle in the d,t-space, but these tangents change when a shortcut is made on a path. This is undesirable because this would mean that all forbidden time objects associated to nodes between the end of a shortcut and any reachable leaves in the graph would have to be updated. The problem of invalidating more d,t than the obstacle becomes more evident when making relatively long edges, since the size of a FT is equal to the length of the edge it is associated with, multiplied by the size of the interval of time at which an obstacle obstructs that edge. Note that there can be multiple obstacles obstructing an edge, I will go into this at a later time. In the case of one obstacle o crossing an edge (x_a, x_b), it is possible to determine the part of the edge which is obstructed at at least one moment by o, (x_c, x_d). Make this into a separate edge with the FT of edge (x_a, x_b) linked to it. If the edge (x_a, x_c) is not of zero-size, make this into a separate edge with an empty set of FTs and do the same for edge (x_b, x_d). The result of this on figure 3(a) can be seen in figure 3(b).

[1]Depending on the speed, the shape of an obstacles in a d,t-diagram will stretch. In this case we consider the obstacles travel at a speed such that they do not appear stretched, even though this would not matter for this purpose.



(a) A d, t-diagram showing a forbidden time-interval.

(b) The result after dividing the edge (x_2, x_{goal}) into three parts.

Figure 3: Before and after the edge segmentation. Notice the reduction of the size of the Forbidden Time-area (tilled square).

4.1.4 Edge Segmentation with Multiple FTs

When multiple obstacles intersect with an edge, x_c becomes the point on the edge which is closest to x_a and is obstructed at least once by any of those obstacles. x_d becomes the point on the edge which is closest to x_b and is obstructed at least once by any obstacle. This does leave one case in which there is a part of the edge(x_a, x_b) which is never obstructed by an obstacle, but lies between two parts of the edge which do get obstructed at some point. Due to time constraints this has not been implemented.

4.2 Rapid Random Tree: TraPeziums

The second method I have developed is also a variation on the RRT algorithm and is called the Rapid Random Tree with TraPeziums. Instead of mapping the obstacles like the RRT-FT does, it maps the valid speeds at which the entity can travel edges.

4.2.1 Idea

The RRT-TP algorithm was designed such that every node in its graph can be reached from the root, while travelling at a constant speed within the speed limitations of e. This is founded on the shortcomings of the RRT-FT algorithm, which does not guarantee a solution when the graph connects x_{start} and x_{goal}.

In figure 1, the relation between the Euclidean dimensions, (x, y), and the path/time dimension, (d, t), is explained. When considering the d, t-space in figure 1(b), and we add an obstacle to a diagram, we can eliminate the part of space that is obstructed by that obstacle. This leaves us with the part of the d, t-space which can be reached at a constant speed (figure 5). The shape of the obstacle is not very relevant at this point, since the construction of the reachable d, t-space is done by taking the two tangents of the obstacle seen from the origin. This is much like the way that a cone of light from a flashlight behaves when casting shadows.

When looking at this d, t-diagram, it can be seen that the d, t-space in which an entity is allowed to travel between two nodes, is shaped like a trapezium (figure 5). The only exception to this is the shape between the start configuration and its successor which is a triangle, which can be thought of as a degenerate trapezium. I will refer to trapeziums which represent a segment of the d, t-space as simply trapeziums. The idea of this method is to store unobstructed pieces of d, t-space as trapeziums with each edge, such that it can be queried for a path that can be travelled at a constant speed. Each trapezium is constructed such that it can be linked to at least one trapezium from the preceding edge until the root (x_{start}) is reached. In order the define when this is case, I first have to define what I call the origin of a trapezium. This is the intersection of the extensions of the non-parallel edges of a trapezium in d, t-space as is seen in figure 4.

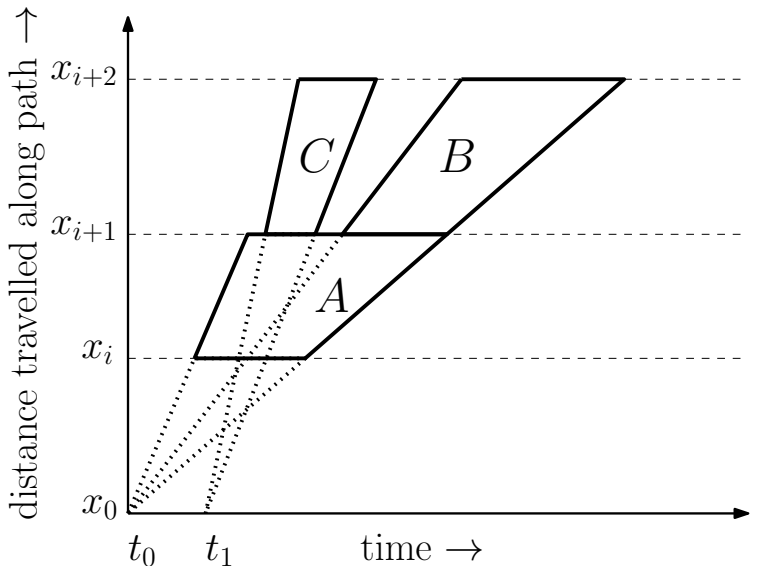


Figure 4: Two trapeziums, A, B, having the same origin (x_0, t_0) and another one, C, with a different origin. The dotted lines are the extensions of the non-parallel edges of the trapeziums.

Next to the origin of a trapezium, I use the abbreviations defined in figure 8 to define the departure times of a trapezium as the interval of time between the time of ld and rd, and the arrival times of trapeziums as the interval of time between the time of lu and ru.

The first requirement for the two trapeziums to be linked, is that the first one, tp_A, associated to the preceding edge of the edge to which the second trapezium, tp_B, is associated. Secondly, the origin of the trapeziums must be the same, and lastly the departure times of tp_A must overlap with the arrival times of tp_B.

Constructing new trapeziums in such a fashion that they can always be linked to a preceding one, gives the advantage that when the tree connects to the goal configuration, it is guaranteed that a solution exists. As mentioned earlier, the disability to do this was one of the shortcomings of the RRT-FT algorithm. I have also discussed the fact that the RRT-FT is really suitable for making shortcuts, which is opposed to the RRT-TP algorithm, since a trapezium-object represents a specific interval of arrival times. Making a shortcut on the graph would change the times at which e can arrive at a node, and thus all nodes that are children of this node might be unreachable when travelling using the new shortcut. For now I only want to point out this complication, as I will discuss the solution for this further on in this section.

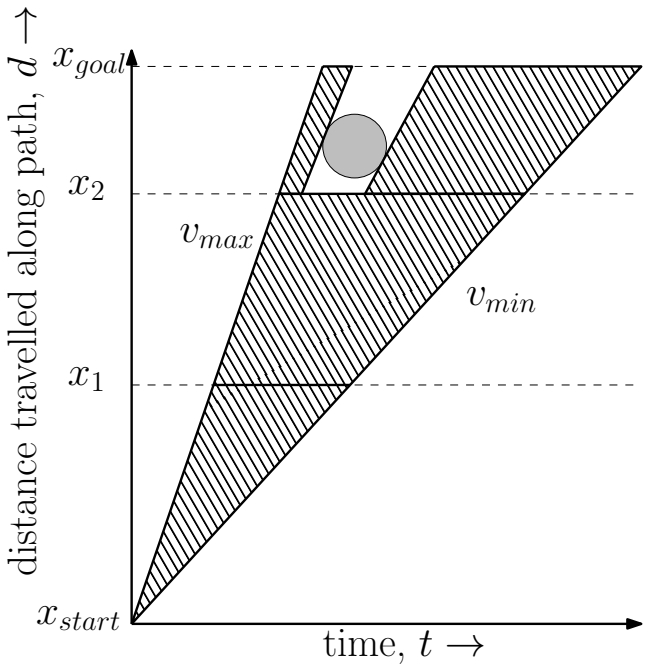


Figure 5: The grey circle is representing an obstacle. All of the d, t-space that is reachable by the entity starting at the origin is highlighted. The entity may only move at a constant speed.

4.2.2 Implementation Details

Adding new edges In this variation on the RRT algorithm, each edge links to a non-empty set of trapeziums. It is possible that given a candidate edge (x_a, x_b), no valid trapeziums exists, in which case, (x_a, x_b) is no longer a candidate and the algorithm moves on to attempting another candidate edge (algorithm 8).

Algorithm 8 EXTEND_TP(Graph τ, Node x)

```
x_near ← NEAREST_NODE(x, τ)
if DISTANCE(x, x_near) > u then
    Move x towards x_near until DISTANCE(x_near, x) = u
x_p ← x_near.predecessor
TPs_in ← the trapeziums associated to edge(x_p, x_near)
for all tp_i ∈ TPs_in do
    TPs_out.ADD(GET_CORRESPONDING_TPS_TO(tp_i))
if TPs_out ≠ empty then
    ADD_VERTEX(x)
    ADD_EDGE(x_near, x, TPs_out)
    if x_new = x then
        return Reached
    else
        return Advanced
return Trapped
```

Finding Valid Trapeziums Given an edge (x_i, x_{i+1}) and its predecessor (x_{i-1}, x_i), it is possible to find the trapeziums that should be linked to (x_i, x_{i+1}) using the trapeziums from (x_{i-1}, x_i) and the obstacles in O. In order to implement this within the time frame of this thesis, I chose to create a sampling method as described in algorithm 9. The idea behind this method is to construct the candidate space, tp_{nc}, which is equivalent to the trapezium that would be constructed if there were not any obstacles. The candidate space is then sampled by stepping through the different departure times of tp_{nc}. These departure times are linked to arrival times, which can be found by mapping the departure time from the interval of departure times to the interval of arrival times. The same result can be achieved by drawing a line line through the sampled departure time and the origin of tp_{nc} and taking the intersection of that line with the arrival times of tp_{nc}. When the departure and the arrival time are known, the sample is constructed and can be checked whether it intersects any obstacles. Valid samples that are subsequent are put together to form a new trapezium. A graphical representation of the sampling can be seen in figure 6.

Algorithm 9 GET_CORRESPONDING_TPS_TO(Trapezium tp, Node x_{i-1}, Node x_i, Node x_{x+1})

$\delta \leftarrow$ the step size for the sampling
$TP_{i,i+1} \leftarrow$ an empty set of trapeziums
define tp_{cs} as a trapezium representing the candidate space
$tp_{cs}.v_{min} \leftarrow \text{DISTANCE}(x_{i-1}, x_i)/(tp.ru - tp.rd)$
$tp_{cs}.v_{max} \leftarrow \text{DISTANCE}(x_{i-1}, x_i)/(tp.lu - tp.ld)$
$tp_{cs}.ld \leftarrow tp.lu$ ▷ [2]
$tp_{cs}.rd \leftarrow tp.ru$
$tp_{cs}.lu \leftarrow tp_{cs}.ld + \text{DISTANCE}(x_i, x_{i+1})/tp_{cs}.v_{max}$
$tp_{cs}.ru \leftarrow tp_{cs}.rd + \text{DISTANCE}(x_i, x_{i+1})/tp_{cs}.v_{min}$
boolean currentIsValid $\leftarrow false$
define tp_{nc} as the next candidate trapezium
for $t_i \leftarrow tp.lu$ **to** $tp.ru$ **step** δ **do**
    $t_{i+1} \leftarrow$ map t_i from interval $(tp.lu, tp.ru)$ to interval $(tp.ld, tp.rd)$
    $v_{i-1,i} \leftarrow \text{DISTANCE}(x_{i-1}, x_i)/(t_i - t_{i-1})$
    $t_{i+1} \leftarrow t_i + \text{DISTANCE}(x_i, x_{i+1})/v_{i-1,i}$
    boolean previousWasValid $\leftarrow$ currentIsValid
    currentIsValid $\leftarrow$ trajectory$((x_i, t_i), (x_{i+1}, t_{i+1})) \cap O = \emptyset$
    if currentIsValid equals previousWasValid **then**
        continue
    else if ($\neg$previousWasValid)$\wedge$ currentIsValid **then**
        $tp_{nc}.ld \leftarrow t_i$
        $tp_{nc}.lu \leftarrow t_{i+1}$
    else if previousWasValid $\wedge(\neg$currentIsValid) **then**
        $tp_{nc}.rd \leftarrow t_i - \delta$
        $tp_{nc}.ru \leftarrow$ map $tp_{nc}.rd$ from interval $(tp_{cs}.ld, tp_{cs}.rd)$ to interval $(tp_{cs}.lu, tp_{cs}.ru)$
        $TP_{i,i+1}.\text{ADD}(tp_{nc})$
    if currentIsValid **then**
        $tp_{nc}.rd \leftarrow t_i - \delta$
        $tp_{nc}.ru \leftarrow$ map $tp_{nc}.rd$ from interval $(tp_{cs}.ld, tp_{cs}.rd)$ to interval $(tp_{cs}.lu, tp_{cs}.ru)$
        $TP_{i,i+1}.\text{ADD}(tp_{nc})$
return $TP_{i,i+1}$

[2] The symbols ld, rd, lu, and ru, are explained in figure 8

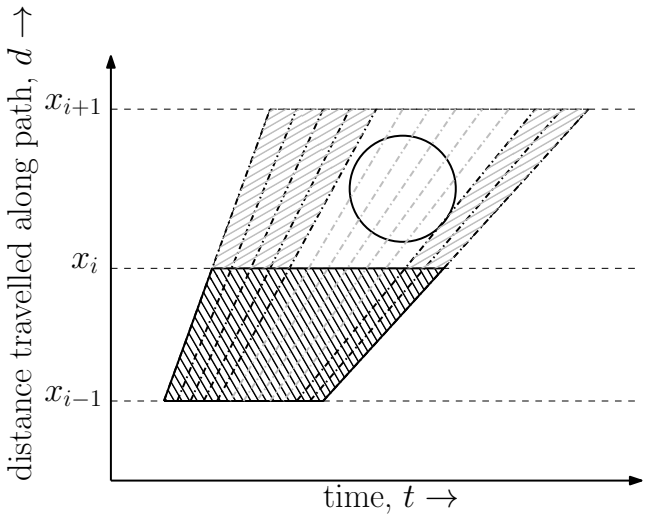


Figure 6: The construction of the set of trapeziums (light gray) which belong with the edge $(x_i x_{i+1})$ from the trapezium belong with the preceding edge (black). The circle represents an obstacle. The dotted-dashed lines represent the samples.

4.2.3 Handling Shortcuts

Given a path which is a solution to the problem stated in section 3.1, $P : \{x_0, x_1, \ldots, x_n\}$, such that $x_0 = x_{start}$ and $x_n = x_{goal}$. Introduce a shortcut from x_i to x_j with $0 \leq i < j \leq n$ to P. This leads to a set of new arrival times on x_j which may not connect to any trapeziums of the edge (x_j, x_{j+1}). All other edges in $\{x_j, \ldots, x_n\}$ may need to be updated if this is the case. To update the trapeziums associated with an edge, more or less the same procedure can be used as seen algorithm 8 except that there now may be more predecessors of x_{near} as a result of the shortcut. When constructing trapeziums from all of these predecessors, it can occur that trapeziums within a the updated set that is associated to an edge overlap.

4.2.4 Joining Trapeziums

When trapeziums within a set, $TP_{i,i+1}$, associated to an edge $ed_{i,i+1} : (x_i, x_{i+1})$ overlap, it becomes more expensive to query that tree or to connect a new edge to x_{i+1} since there are more preceding trapeziums. In order to counter this, trapeziums are joined together into a new trapezium when this is possible. However, this is not possible for all trapeziums within a set, since the conjugation of two overlapping trapeziums is not per se a trapezium, thus not all overlapping trapeziums are suitable for joining. In figure 7 two pairs of overlapping trapeziums are shown which are not suitable for joining. For a pair of trapeziums, (tp_A,tp_B), to be joined, it is required that the following is true for that pair[a];

$$tp_A.ld \leq tp_B.ld \leq tp_A.rd \wedge tp_A.lu \leq tp_B.lu \leq tp_A.ru$$

$$\vee$$

$$tp_B.ld \leq tp_A.ld \leq tp_B.rd \wedge tp_B.lu \leq tp_A.lu \leq tp_B.ru$$

[a]The abbreviations lu, ru, ld, rd are shown in 8

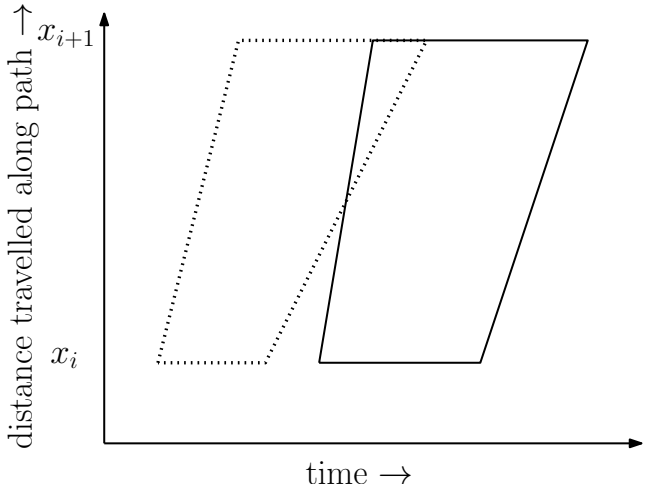


Figure 7: An example of two overlapping trapeziums that are not suitable to be fused into one new trapezium.

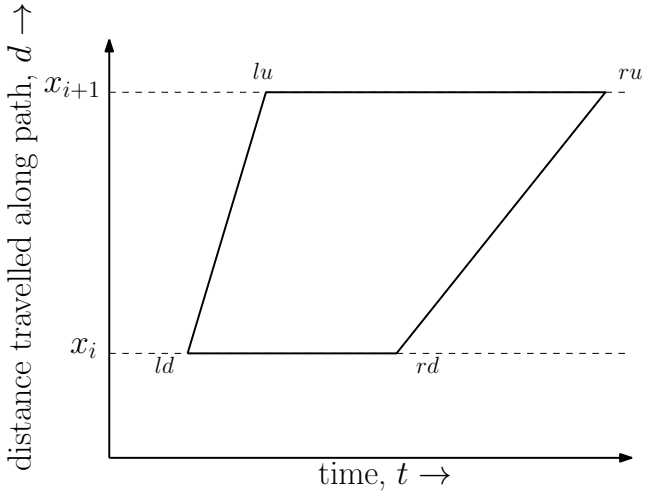


Figure 8: The names of the corners of a trapezium. The letters stand for the position (left/right/up/down).

4.3 Rapidly-exploring Random Tree

The original RRT algorithm has several parameters of itself which I will explain in this subsection. Next to that, I will discuss the methods which apply to both the RRT-FT and the RRT-TP algorithms that are required to conduct the experiments.

4.3.1 Introducing Goal Bias

Path planning problems can be classified into two categories, the ones which require the entity to end in an exact state, and those which require the entity to end in a region or a set of states. In this thesis we are dealing with a problem of the first category, since e should end up at exactly x_{goal}. This means that the approach that is used to solve the problem must be targeted to reach the goal state. One way to accomplish this when dealing with RRTs, is to introduce a bias in the sampling of the random nodes towards the goal configuration (algorithm 10). This method requires a parameter, g, to set the probability that x_{goal} is attempted to be connected to the graph. After running a pilot experiment, it became clear that using $g = 0.05$ is a reasonable value, since the average number of nodes before x_{start} and x_{goal} became connected was roughly 45 for both algorithms during the same pilot experiment. Using $g = 0.05$ means that on average the goal state is attempted to be added to the graph every 20 nodes. This value seems balanced compared to the 45 nodes that are added on average before x_{start} and x_{goal} become connected, because it suggests that on average the goal state is attempted once without success before the goal state is added with success. Having this succeed in half of the times seems appropriate, even though it is hard to say exactly why this ratio seems balanced.

Algorithm 10 BuildRRT(GoalBias g, Node x_{start}, Node x_{goal})

```
1: τ ← a graph containing only x_start
2: while ¬GoalIsReached do
3:     if g > RandomNumberBetween(0, 1) then
4:         AttemptGoal(τ,x_goal)
5:     else
6:         x_rand ← GetRandomConfiguration()
7:         AddNode(τ, x_rand)
```

4.3.2 Maximum Edge Length

A difference between Rapid Random Trees and Probabilistic Roadmaps (PRM) is that the RRT does not implement a form of greedy exploration, while PRM does. This is done by introducing u, the parameter for the maximal edge length, which can be as seen in algorithm 2. By limiting the length of an edge, the probability of successfully adding an edge increases as is discussed in [9]. Choosing a value for u depends for the most part on the obstacles in the scene and the distance between the start and the goal state. From a pilot experiment with visual inspection, I found that $u = 20$ yields edges which are in proportion with the scene.

4.3.3 Query Method and Solution Selection

I chose to compare solutions by their length, since this is an insightful way of measuring the quality of a path. The trees were queried by using a depth-first search, which was still applicable after shortcuts were made in the graphs, since the graphs are directed. The motivation behind this quality measure is given in section 3, and as mentioned before all distance metric are strictly Euclidean.

4.3.4 Shortcut Strategy

As discussed in section 3.2, an important requirement for the developed methods was that they allowed for shortcuts. After the start and goal configuration become connected, shortcuts are made as described in algorithm 11. The idea of this method is to attempt a certain number of shortcuts between the nodes of the solution, followed by another query for a solution on the tree. By not updating the solution after a shortcut, all possible combinations of shortcuts are considered during the second query. In the experiments that were conducted, the number of shortcuts that are attempted was set to 100. This value was chosen because pilot experiments showed that the average number of nodes in the solutions was around 10, which makes that 100 is a number that should give a good idea of how well shortcuts in general improve the quality of the solution.

Algorithm 11 MakeShortcuts(Graph τ, Path $P : x_0, \ldots, x_n$)

```
1: for nrAttempts ← 0 to maxNrAttempts do
2:     repeat
3:         a ← RandomIntegerInRange(0, n)
4:         b ← RandomIntegerInRange(0, n)
5:     until |a − b| > 2
6:     c ← min(a, b)
7:     d ← max(a, b)
8:     AttemptEdgeBetween(x_c, x_d, τ)
```

5 Experimental Setup

In the previous section, the RRT-TP and the RRT-FT algorithms were discussed, together with the implementation details of these methods. Next to this, I had to make a number of choices considering the implementation of the experiments. In this section, I will go into the details which are not per se part of the algorithms, but are essential for the experiments. Before that, I will discuss the method that I will use to answer the research question, in other words the goal of the experiments.

5.1 Goal of the Experiments

As stated in section 3.2, the main goal of this research is to determine whether the RRT-FT and the RRT-TP methods are suitable for planning paths which can be travelled at a constant speed. In order to determine this, I will measure the rate of success of both methods and compare them with each other. The rate of success is defined by the ratio between the number of runs in which the RRT finds a valid solution and the total number of runs of that RRT.

To answer the second question; "Which algorithm produces shorter paths on average?", and the third question; "How does making shortcuts influence the length of the shortest path?", the lengths of the shortest solutions will be measured before and after shortcuts have been made, and these measurements will be averaged.

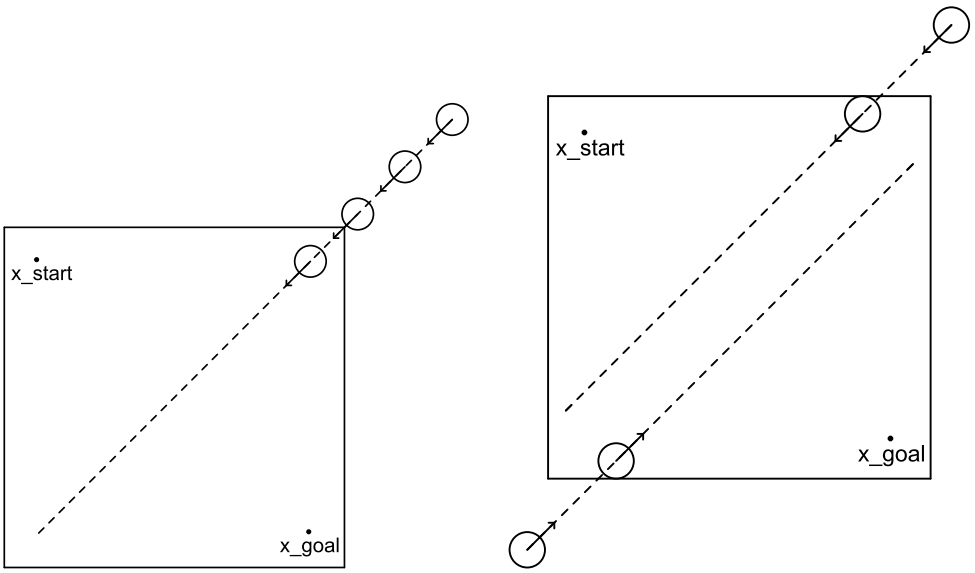
For the question; "How does the averaged time it takes to make a solution compare between the new algorithms?", the time spent on building the tree will be measured and averaged, to see which of the two algorithms is faster on average.

5.2 Scene Design

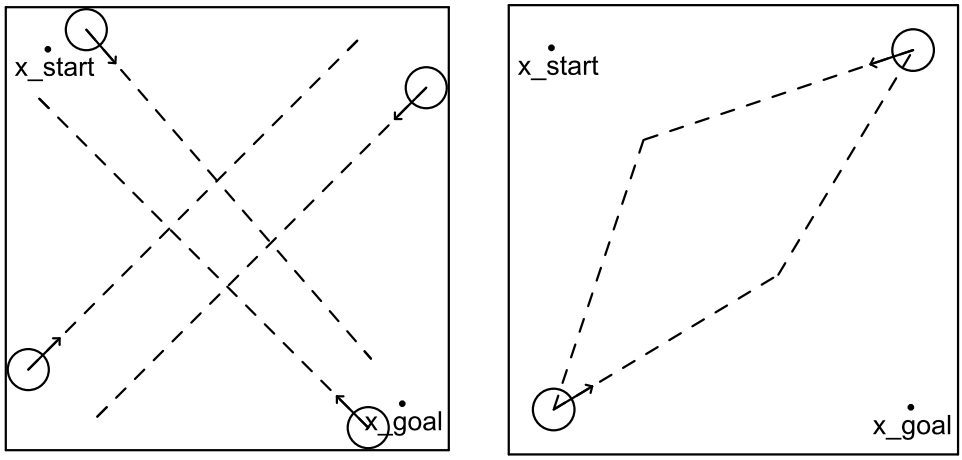
The scenes were designed in such a manner that they represented situations similar to those seen in Air Traffic Management, even though the RRT-TP and the RRT-FT algorithms were not designed specifically for this type of situation. An example of one of these ATM influenced situations is a scenario called *String*, where the moving obstacles travel in a line right behind each other (figure 9(a)). This comes from the fact the aircraft typically travel along set paths at roughly the same speed. Another example of this, is the scenario called *Guillotine*, which is similar to *String* except that the aircraft are coming from two directions. The other two scenarios, *ImplodeTilt* and *Diamond*, are not directly based upon an ATM situation, but are designed to be challenging scenarios that are different from the first two. Namely, *Diamond* has moving obstacles that do not travel in a straight path and *ImplodeTilt* is more crowded by moving obstacles, especially around the center of C_{room}.

All parts of the scenarios are separately described in xml-files and allow for easy modification and expansion of the experiments. Each obstacle set has been given a name based on the shape of the paths of the obstacles that are part of that set. All obstacles have a known trajectory (2D+T) and travel at a constant speed between the points in that trajectory. The times in these 2D+T trajectories were chosen such that the obstacles travel at 1 unit of length per second.

The space, C_{room}, in which the entity is allowed to move has a Euclidean dimension of $[100 \times 100]$ with the start configuration at $(10, 10)$ and the end configuration at $(90, 90)$. These dimensions are based on the situations that were dealt with during the C-Share project and the same goes for the proportion between the number of obstacles and the size of the configuration space. The entity begins at the start configuration at $t = 0$. The speed limitations of e are $v_{min} = 0.5$ and $v_{max} = 2.5$.



(a) *String* (b) *Guillotine*



(c) *ImplodeTilt* (d) *Diamond*

Figure 9: The four scenarios that were used for the experiments. The arrows indicate the direction in which the aircraft are going.

5.3 Technical Details

All experiments were implemented in C++ using Microsoft Visual Studio express 2010. The following libraries were used;

std/stl Where possible I have used the standard C++ libraries.

Atlas This library was used to manage the graphs of the RRT algorithms.

Callisto A 3D visualizer library also suitable for collision detection. This was only used as a visualizer in order to validate the solutions (figure 16,17).

All experiments were conducted on a Intel Celeron M CPU 410 1.46 GHz.

6 Results

In this section I will present the data obtained from the experiments as discussed in section 5.1. First of is the rate of success which has been measured for each scene for both the RRT-FT and the RRT-TP method. Secondly I will present the measurements of the average length of the shortest solutions, followed by the data on the time it took to build the graphs of both of these methods.

6.1 Rate of Success

As described in section 5.1, the rate of success is defined by the ratio between the number of runs in which the RRT finds a valid solution and the total number of runs of that RRT. The data on the number of runs in which a solution was found can be seen in table 1.

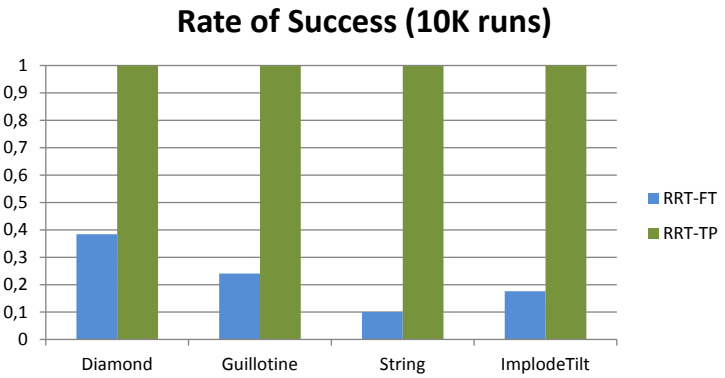


Figure 10: The Rate of Success measured over 10.000 runs.

	RRT-FT	RRT-TP
Diamond	3829	9997
Guillotine	2408	9994
String	1004	9981
ImplodeTilt	1763	9998

Table 1: Number of runs in which a solution was found (10K runs).

6.2 Average Path Length

The average path length is calculated by taking the average of the length of the shortest path of every run that produces a valid solution. The measurements take place before and after shortcuts have been made.

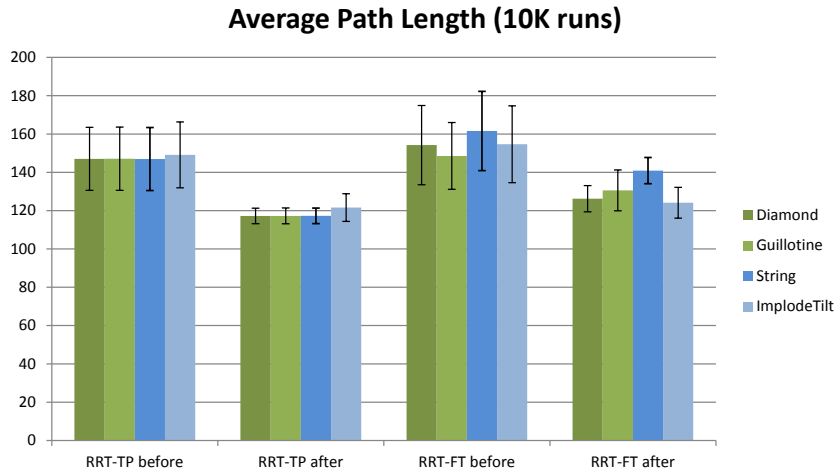


Figure 11: The average path length was measured by averaging the length of the best solution of every run, over 10.000 runs. The error bars indicate two standard deviations.

6.3 Average Build Time

In this subsection the data concerning the average build time of the RRT-FT and the RRT-TP are put forward. I have managed to obtain enough data so that I was able to use statistics to show that the build time of both algorithms can be modeled with a log-normal distribution. This was discovered after I had attempted to describe the data with a normal distribution, but since the standard deviation was in the same order of magnitude as the mean, I suspected that a normal distribution would not be accurate. Another reason why a log-normal distribution in this case should be preferred over a normal distribution, is because the build time of a graph can not be less than zero seconds, which agrees with a log-normal distribution opposed to a normal distribution. I have analyzed all 8 cases (4 scenarios times 2 algorithms) by looking at the graphs of (1) the calculated log-normal distribution and (2) the relative frequency scores of the build times, of which an example is shown in figure 15. This showed in all cases to be an accurate model, therefore I presume that the log-normal distribution should be applied. Also, the Chauvenet criterion [11] was used in order to determine the number of outliers, which was in each case less than 0.1% of the measurements, which implies that the probability function that is used as a model applies to $> 99.9\%$ of the data.

In the figures below, three graphs are shown which visualize the probability densities of the build time of the algorithms. Since the RRT-FT and the RRT-TP score very differently, I have chosen to display each graph twice; once in the overview of all build time measurements and once separate from the other algorithm to allow for more detail.

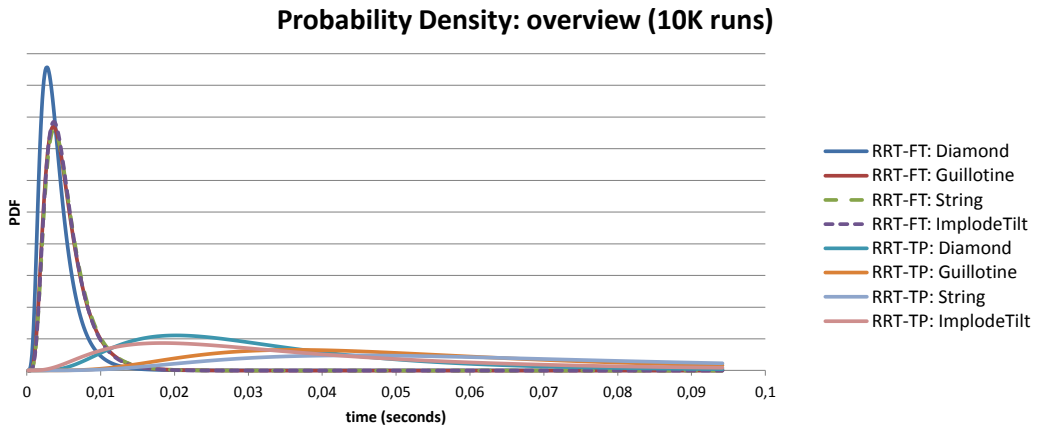


Figure 12: The probability densities of the building times of both the RRT-FT and the RRT-TP algorithms. The following curves of RRT-FT are overlapping; *ImplodeTilt*, *Guillotine* and, *String*. Since this a probability density, the y-axis shows the relative probability.

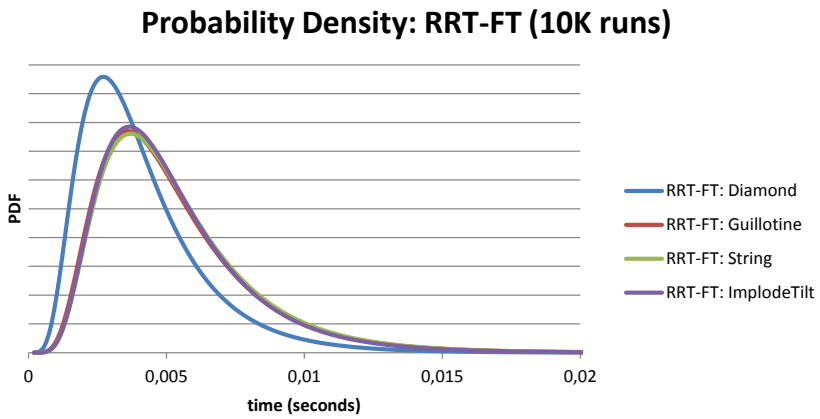


Figure 13: A graph similar the one shown in figure 12. The probability densities of the building times of the RRT-FT algorithm. *Guillotine* is overlapping with *String* and *ImplodeTilt*.

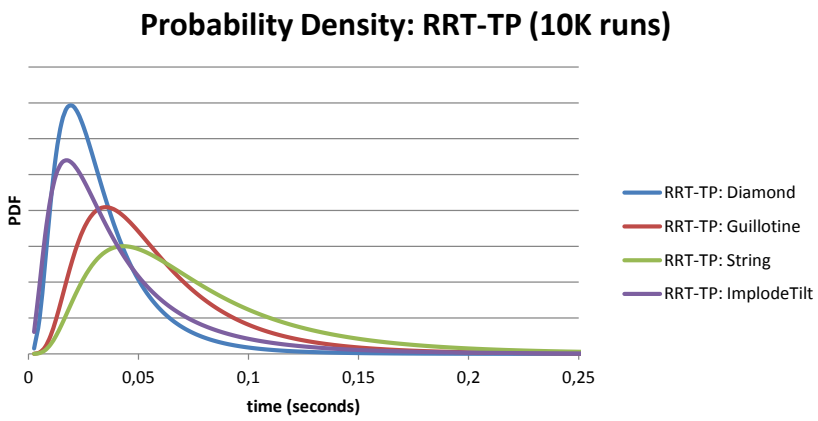


Figure 14: A graph similar the one shown in figure 12. The probability densities of the building times of the RRT-TP algorithm.

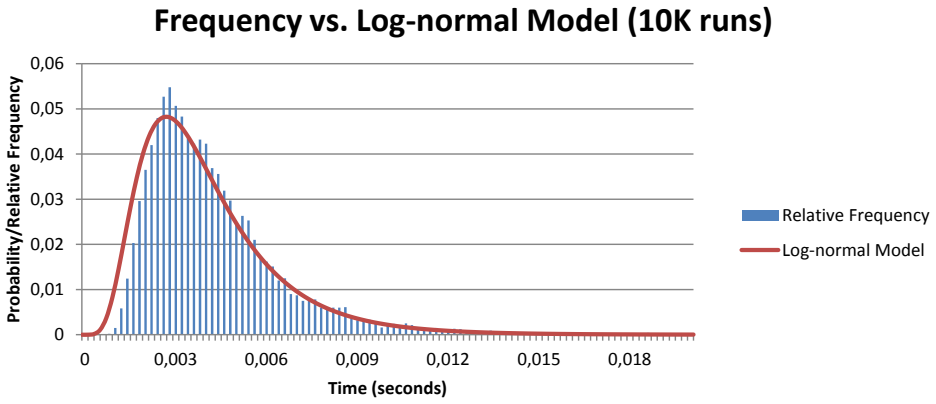


Figure 15: This graph compares the relative frequencies of the build time of the RRT-FT (blue), with a log-normal probability function (red). The build times were measured over 10K runs and scenario *Diamond* was used for this example.

6.4 Graphical Representation

Since the experiments involved three dimensions, two Euclidean and time, it is possible to make a graphical representation of both of the trees with their associated attributes.

6.4.1 RRTTP

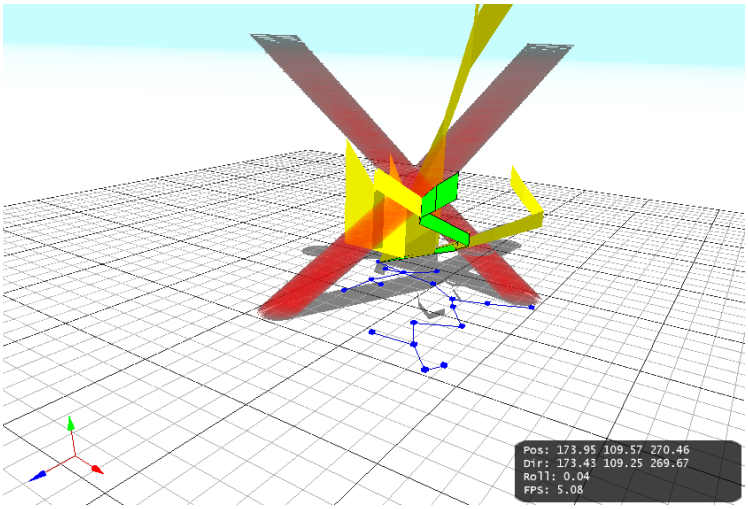


Figure 16: A graphical representation of the tree of the RRTTP method together with the obstacles (red), trapeziums (yellow), Euclidean graph (blue) and solution (green). The axis pointing upwards is used as the time-dimension.

6.4.2 RRTFT

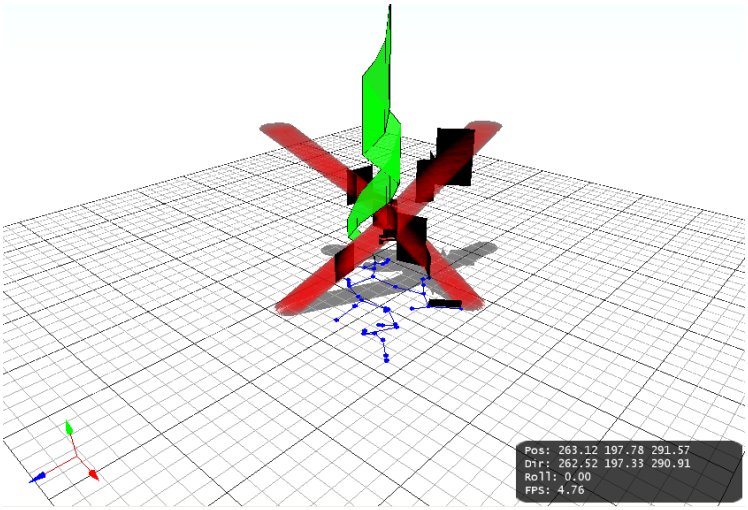


Figure 17: A graphical representation of the tree of the RRTFT method similar to the one shown in figure 16 with the ForbiddenTime object shown in black.

7 Conclusion

In section 3.2 I posed the question whether the RRT algorithm can be adapted in such a way that it will be able to produce paths which can be travelled at a constant speed within the constraints stated in section 3. In the sections after that, the RRT-TP and the RRT-FT algorithms were presented as possible solutions for the problem stated in section 3.1. The result of the experiments in section 5 were presented in section 6, and from these I will answer the research questions in this section.

7.1 Research Question

The main research question for this thesis as posed in section 3.2 is; " Can the RRT algorithms be adapted so that is able to produce solution existing out of a path, P_{2D}, and an interval of speeds, V, such that P_{2D} can be travelled with a constant speed $v_i \in V$, while avoiding moving obstacles with a known trajectory? " When designing possible candidate algorithms, I quickly noted that searching for a solution with a set speed will directly eliminate possible solutions, making it harder or even impossible to find a solution. This is why both algorithms that were designed use the opposite strategy; that is to look for path with an interval of speed at which that path can be travelled. In this sense, the algorithms both present collections of solutions instead of one solution, which was not meant to be the aim of the algorithm, although it does yield more valuable solutions.

To answer the research question, I have measured the rate of success, which is the ratio between the number of runs in which the algorithm was successful and the total number of runs. Note that this is a normalized measure. In figure 10 and table 1 the measurements of the rate of success are displayed, showing that both the RRT-FT and the RRT-TP algorithms have a rate of success that is higher than zero, which means that they are able to find valid solutions in at least some of the runs.

From these measurements it can be concluded that the RRT-FT algorithm has the highest average rate of success for scenario *Diamond*, which is also the scenario containing the smallest number of moving obstacles (figure 9). The other scenarios have a lower rate of success, which is likely due to their complexity. *String* has the lowest rate of success, which can be explained by looking at the obstacles (5.2) that shows that every solution will have to cross paths with four moving obstacles. This means that every solution will contain an edge which has four Forbidden Time objects associated to, making it difficult for the RRT-FT to plan a path. It seems that the RRT-FT has more difficulty dealing with edges with more Forbidden Time objects associated to it, opposed to edges with less Forbidden Time objects, though this can not be said with much certainty from these measurements.

The RRT-TP finds a solution almost every time, while the RRT-FT only succeeds some of the times, depending on the scenario. From this is concluded that both algorithms are variations of the RRT that are suitable for finding paths that can be travelled at a constant speed amongst moving obstacles. Even though the RRT-TP definitely shows to have a higher rate of success for the scenarios that were tested.

7.2 Quality Performance and Shortcuts

In section 5.1, next to measuring the rate of success as a way to answer the main research question, I have also discussed measuring the length of the solutions in order to answer the questions; "Which algorithm produces shorter paths on average?", and; "How does making shortcuts influence the length of the shortest path?". The results of these measurements can be seen in figure 11, from which can be concluded that RRT-TP on average gives shorter paths than RRT-FT, both before and after shortcuts have been made. It can also be concluded that making shortcuts reduces the spread in path lengths. The difference in the

average length of the shortest paths between the RRT-FT and the RRT-TP algorithms can be explained by the fact that the RRT-FT invalidates more d, t-space than is needed, due to the nature of the algorithm (section 4.1.3), opposed to the RRT-TP. This implies that the RRT-FT has a smaller probability of planning a path through a narrow passage, which limits its possibilities with respect to the RRT-TP, eventually leading to the fact that the RRT-FT has a lower probability of finding shorter paths. The scenarios do not allow this conclude for the general case this with much certainty, however it can be concluded that the RRT-TP produces shorter paths that the RRT-FT for these scenarios.

7.3 Build Time

The third aspect that was measured is the time it took both algorithms to build the graph that should answer the last research question; " How does the averaged time it takes to make a solution compare between the new algorithms? ". As discussed in section 6.3, the data was shown to be log-normally distributed, hence the measurements were presented in the form of graphs of the probability densities. In these graphs (figures 12, 13, 14) it can seen that the RRT-FT has its probability densities closer to zero than the RRT-TP, which means that on average the RRT-FT takes less time to build a graph. This is further discussed in the Discussion & Future Work section (section 8).

The measurements of the average build time shows some resemblance to the measurements of the average length of the shortest paths, since *Diamond* is also showing to require a smaller build time compared to the other scenarios. This can be seen in the graphs showing the probability densities, as *Diamond* has more probability that is closer to zero than the other curves. In this case of RRT-FT it is easily explained why *Diamond* has a shorter build time than the other scenarios, *Diamond* contains two obstacles while the other scenarios contain four obstacles. The RRT-FT builds the graph in the same fashion every time it is ran regardless of the scenario, because it does not take into account whether a node can be reached from the root and thus 'blindly grows on'. The only noticeable difference between the scenarios for the RRT-FT, when it comes down to build time, is the number of obstacles in the scenarios, which is reflected in figure 13.

It is difficult to draw a solid conclusion about what exactly causes the difference in the average build times between the different scenarios when it comes down to the RRT-TP. However, it can be said that the RRT-TP needs the most time, on average, to build a graph for *String*, which is also the scenario with the lowest rate of success when using the RRT-FT. This is an indicator that the build time is related to the complexity of the scenario, even though this research does not provide enough data to make conclusion about what exactly makes a complex situation, and whether this is the same for both algorithms. It should however be clear that the number of obstacles is an important factor for the build time.

Though it is difficult to draw any conclusions concerning the time it takes to build the graph for the general case, it can be said that the amount of time that is consumed is acceptable when applying the algorithms to the context of ATM. As discussed in section 5.2, the scenarios that were designed as well as the obstacles and the entity are influenced by the ATM context, especially the proportions between the distance between x_{start} and x_{goal}, the size of C_{room} and the size obstacles. Since this is the case, the time it takes the RRT-TP and the RRT-FT to build the graph for the scenarios can be thought of as an indicator for how much time building the graph would take in real situations. In ATM a look-ahead time of 20 minutes is often used, which means that conflicts are dealt with around 20 minutes before the actual violation of the separation criteria would occur. When comparing this look-ahead time with the time it takes to build the graph for the RRT-FT or the RRT-TP, it can be said that both these algorithms consume an acceptable amount of time, and since the scenarios are designed based on the ATM context future investigation is worthwhile.

8 Discussion and Future Work

8.1 Obstacles

In this thesis, all planning was done amidst moving obstacles instead of also using stationary obstacles. This was done in order to show that the RRT-TP and the RRT-FT algorithms work in ATM-like situations where there are no static obstacles. However, ATM does frequently have to deal with weather conditions such as thunderstorms, or more permanent boundaries such as military airspace, which is off limits for normal aircraft. These types of obstacles or boundaries are static, albeit that some are only temporary. Static obstacles can be thought of as dynamic obstacles remaining in one place, thus both of the algorithms that I have designed still apply, even though that it might require different parameters. For instance, when static obstacles form a narrow passage, a smaller value for u should be used to increase the probability of planning a path through this passage. For future work, I recommend that both algorithms should be tested in situations containing both static and dynamic obstacles, in order to show that they still function in those situations.

From the measurements of the rate of success, it also seems relevant to investigate the behavior of the RRT-TP algorithm with more obstacles, because the RRT-TP was equally successful in all four scenarios. Testing the RRT-TP algorithm on scenarios with more moving obstacles or more obstacles in general, will give a better understanding of the limitations of this algorithm.

8.2 Measuring the Path Quality

In this thesis, I have used the simplest way of measuring the path quality that I could think of; the length of path. However, there are lots of aspects to the solutions which one might want to measure and optimize. Examples of this are the size of the interval of flyable speeds related to a path, or the number of nodes in the path. During the experiments, I have measured both these things in order to give an indication on the performance when it comes down to other quality measures. Note that the experiments were implemented such that the shortest path would be selected as the solution, whereas these two values were not taken into account. Secondly, the runs in which no valid solution was produced were omitted from these results. Thirdly, it should be noted that the speed limitations of e are $v_{min} = 0.5$ and $v_{max} = 2.5$ (section 5.2) and the size of the speed interval V is calculated by subtracting the lowest value in V from the highest. For example, in the case that there are no obstacles, V would become $[0.5, 2.5]$ and the size of the speed interval would become 2.

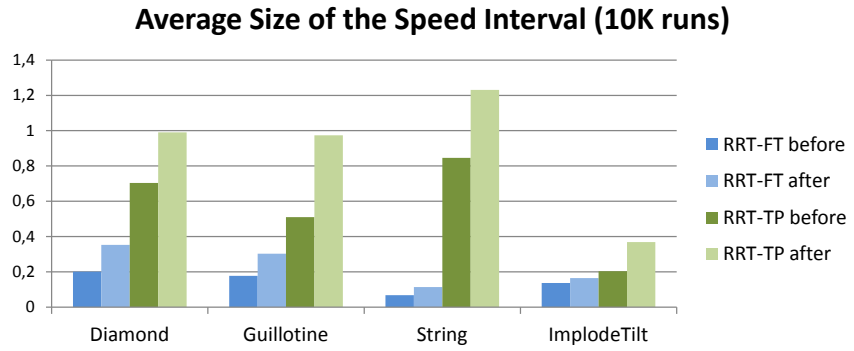


Figure 18: The average size of the speed interval, V, measured over 10K runs.

From these two graphs, it can be concluded that the RRT-TP performs better than the RRT-FT at both these quality measures and in all scenarios. It makes sense that, on

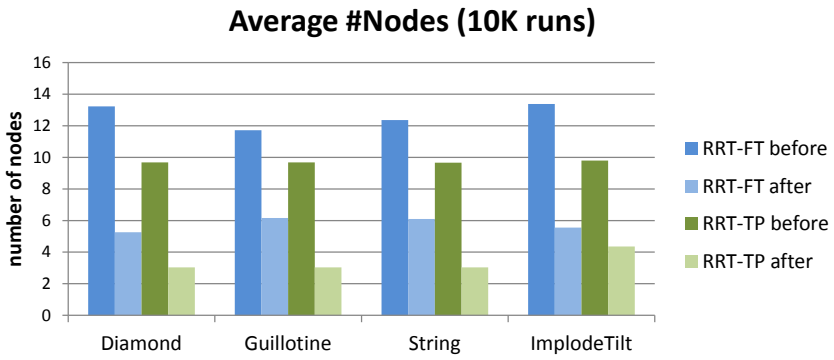


Figure 19: The average number of nodes in a valid solutions, measured over 10K runs.

average, the RRT-TP yields larger speed intervals than the RRT-FT, since it is designed to find exactly all flyable speeds, opposed to the RRT-FT algorithm which invalidates more d, t-space (more details in section 4.2)

What is remarkable about these graphs, is that in each case the average of the size of the speed interval grows by making shortcuts. I suspect this to be caused by the fact that the path of most moving obstacles in O is perpendicular to a line segment from x_{start} to x_{goal}. Making shortcuts and selecting the shortest path, will cause the solutions to converge to this virtual line segment. In turn, this will minimize the size of the intersection between the solutions and the moving obstacles, and thus lead to an increase of the size of the speed interval. The scenario containing the moving obstacles with the largest variety in direction, *implodeTilt*, has the smallest increase in size of the average speed interval as a result of shortcut, which agrees with the previously stated theory. Unfortunately this research does not provide enough data to make any conclusion about the correctness of the theory with any certainty. However, it can be said that RRT-TP has outperformed RRT-FT in this research when it comes to the average size of the speed interval.

When it comes down to the average number of nodes of the solution, the RRT-TP also performs better than the RRT-FT. It can be seen in figure 18 that the RRT-TP algorithm on average needs the same number of nodes in each scenario before the shortcuts have been made. While after the shortcuts are made, *ImplodeTilt* seems to need more nodes than the other scenarios. This likely due to the fact that in this scenario, there are two moving obstacles, which are close to the virtual line going through x_{start} and x_{goal}, which might require solutions to be more evasive which in turn requires more nodes. On the contrary, the RRT-FT algorithm yields a more constant average number of nodes across the scenarios, which is likely due to the fact that it does not take into account whether a node can be reached from the root when taking the criteria from section 3.1 into account.

8.3 Sampling in RRT-TP

In this thesis I concluded that my implementation of the RRT-TP algorithm is slower than the one of the RRT-FT algorithm (section 7.3). Contrary to the RRT-FT algorithm, the RRT-TP algorithm uses a form of sampling in order to determine which parts of the d, t-space are unobstructed. This might be a part of the cause that it runs slower than the RRT-FT. By replacing the sampling with an algebraic method, the running time of the RRT-TP implementation can potentially be shortened and might even outperform the RRT-FT. This will be an interesting aspect to look into in future work.

8.4 Context

For the research part of this thesis, I have put the ATM context of the C-Share in the back of my mind, and focused on making generally applicable methods that are suitable for solving an abstract problem as is formulated in section 3. However, it should not be forgotten that the origins of the developed methods lie within the ATM context, and therefore I would like to advise that in future work, the RRT-FT and the RRT-TP algorithms are tested within the ATM context. I expect to see a great difference with the speed profiling method that was implemented in the C-Share project (section 2), in the sense that the RRT-FT and especially the RRT-TP will provide solutions that are shorter and can be travelled with a much wider interval of speeds. I also expect them to be faster at producing solutions.

When thinking ahead even more, the RRT-FT and the RRT-TP should be tested in higher dimensions such as 3D+time. At this moment aircraft are restricted to discrete flight levels of 1000 feet, implying that the added dimensions is not same and less complex than the first two euclidean dimensions. However, this might subject to change in the future, and at this it is still necessary to maneuver between these flight levels which also requires the third dimension to be continuous instead of discretized. The RRT was chosen as a foundation of both methods because it is known to handle higher dimensionality well, which was expected to be required in not only the C-Share project, but also in other applications. Having an extra Euclidean dimension will not invalidate the applicability of these methods, because the d,t-space will always have two dimensions. This is because all Euclidean dimensions are captured in one dimension, d, when a candidate edge is created, thus making it applicable to Trapeziums and Forbidden Time objects. Even when dimensions such as for instance the pitch and the roll of an aircraft are involved, the RRT-FT and the RRT-TP will still hold, even though it will become much harder to visualize.

A third point of interest within the ATM context, is to add curved or otherwise non-linear trajectory segments as edges to the graphs of the RRT-FT and the RRT-TP. This should be possible because the Euclidean dimensions of one edge are captured in one dimension, d, as explained before. In this C-Share project it was chosen to simplify the trajectories to be made up out of line segments with the assumption that an aircraft can have instantaneous acceleration. In order to be able to reach a level of accuracy required for planning paths for ATM applications, non-linear edges should be added and tested to the RRT-FT and the RRT-TP.

Finally, it would be interesting to look into ways for replanning two aircraft simultaneously. At this moment, the new algorithms only allow planning for one entity at a time. It is likely possible to plan paths for two entities using the RRT-TP, since the trapeziums of the first entity would become obstacles for the second entity. In this way, both entities can receive a new path with both their own speed interval V as was the case with one entity. However, this is still a form of prioritized planning, which becomes clear when considering the situation where two edges from different entities intersect in the euclidean dimensions. Namely, the trapeziums associated to these edges have to be constructed in an order, because the trapeziums associated to the first edges have to be fully constructed before the trapeziums associated to the second edge can take them into account. Switching the order in which the trapeziums for one or the other entitity are constructed influences the size of the trapeziums, since the entity that goes first will not have to take the second entity into account. This might be solved by linking trapeziums from different entities that intersect each other, and allowing these trapeziums to change their size in responds to the size of the trapezium that it is linked to. The result of this would be that the speed intervals of the solutions for the different entities would be linked as well, meaning the size of V of one solution might be reduced in order to benefit the size of V of the other solutions. This idea will likely become complex rather fast when the number of intersects, and thus the linked trapeziums, increases, which might mean that this solution is best applied to problems with

two entities. Altogether I would recommend the RRT-TP for further research on planning paths for two entities simultaneously.

Adapting the RRT-FT to avoid prioritized planning is much harder, because the times at which an entity travels over an edge are not stored with it. This means that when one entity has found a solution, it becomes an obstacle for the second entity and the sets of forbidden time objects associated to the edges in its graph must be updated. This way of planning is not expected to yield different results than it would have with prioritized planning.

References

[1] L. Jaillet, J. Hoffman, J. Van den Berg, P. Abbeel, J.M. Porta, and K. Goldberg. EG-RRT: Environment-guided random trees for kinodynamic motion planning with uncertainty and obstacles. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 2646–2652. IEEE, 2011.

[2] M. Kalisiak and M. van de Panne. RRT-blossom: RRT with a local flood-fill behavior. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1237–1242. IEEE, 2006.

[3] R. Kindel, D. Hsu, J.C. Latombe, and S. Rock. Kinodynamic motion planning amidst moving obstacles. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 537–543. IEEE, 2000.

[4] J. Kosecka, C. Tomlin, G. Pappas, and S. Sastry. Generation of conflict resolution manoeuvres for air traffic management. In *Intelligent Robots and Systems, 1997. IROS'97., Proceedings of the 1997 IEEE/RSJ International Conference on*, volume 3, pages 1598–1603. IEEE, 1997.

[5] J.J. Kuffner Jr and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[6] F. Lamiraux, E. Ferré, and E. Vallée. Kinodynamic motion planning: connecting exploration trees using trajectory optimization methods. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3987–3992. IEEE, 2004.

[7] S.M. LaValle. Rapidly-Exploring Random Trees A New Tool for Path Planning, Oct. 1998. TR 98-11, Computer Science Dept., Iowa State Univ. http://janowiec.cs.iastate.edu/papers/rrt.ps.

[8] S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[9] S.M. Lavalle and J.J. Kuffner Jr. Rapidly-Exploring Random Trees: Progress and Prospects. In *Algorithmic and Computational Robotics: New Directions*. Citeseer, 2000.

[10] G.P. Roussos, D.V. Dimarogonas, and K.J. Kyriakopoulos. 3D navigation and collision avoidance for a non-holonomic vehicle. In *American Control Conference, 2008*, pages 3512–3517. IEEE, 2008.

[11] J.R. Taylor. An introduction to error analysis: the study of uncertainties in physical measurements. Univ Science Books, 1997.

[12] C. Tomlin, G.J. Pappas, and S. Sastry. Conflict resolution for air traffic management: A study in multiagent hybrid systems. *Automatic Control, IEEE Transactions on*, 43(4):509–521, 1998.

[13] C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1178–1183. IEEE, 2003.

[14] J. Venema. Using Robot Path Planning techniques in Air Traffic Management. Master's thesis, Faculty of Aerospace Engineering , Delft University of Technology, 2009.

9 Acknowledgements

Firstly, I would like to thank my supervisor Dennis Nieuwenhuisen for being a great supervisor and a great friend. Next, I would like to thank Marc van Kreveld for setting up my internship at the NLR and for all his criticism on my writing. I would like to thank everybody at the Air Traffic and Airport department of the NLR (ATAP) for having me and being so friendly. I really enjoyed learning about ATM and all of the cool stuff you guys are doing. I want to thank Pim van Leeuwen, Tanja Bos and Rolf Klomp for their collaboration on the C-Share project, I really enjoyed our time together. I would like to thank Rene, Guido and Lisa for their pleasant company.