UAV 4D Path Planning using Rapidly-exploring Random Trees

Manuel Rosa
manuel.d.rosa@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Portugal

November 2018

Abstract

Developments in the aviation industry have led to an increase in air traffic, requiring all operations to be more flexible and efficient without disregarding safety standards. Simultaneously, the amount and variety of real-time data available to air transport systems and their crews are also increasing. This situation calls for higher levels of automation where humans in the loop focus on their supervising role. For this reason, a Smart Auto-Flight Control System (SAFCS) is being developed to continuously plan, execute and monitor airborne missions in which it is involved. It uses digital communications to interact with other agents and applies path planning algorithms to find the best trajectories for each scenario. The Rapidly-exploring Random Tree (RRT) is one of such path planning algorithms, first introduced as being capable of dealing with high-dimensional, nonholonomic systems with kinodynamic constrains. Such characteristics have led to widespread applications and the continuous development of RRTs over the past years including extensions of these algorithms to satisfy anytime, dynamic and online requirements. This dissertation integrates the RRT planning family into the SAFCS project, discussing its benefits and selecting the algorithms that better fit the requirements. In certain cases, enhancements to existing algorithms are proposed to solve specific problems. Finally, the implementation is evaluated and results are presented.
Keywords: Path Planning, Sampling Based Planning, RRT, SAFCS

1. Introduction

Aviation is a sector where newer and better solutions are always expected in order to solve challenging problems continuously arising. The sky, once considered as free and unregulated space where anyone could fly as desired, is now strictly controlled to maintain safety in an increasingly congested environment. Over the past decades, developments in many different engineering fields have led to an increase in the number and variety of aerial vehicles, thus requiring all operations to be more flexible and efficient without disregarding safety standards. Simultaneously, the amount and variety of real-time data available to air transport systems and their crews are also increasing. This situation calls for higher levels of automation where humans in the loop focus on their supervising role.

Having identified the current state of the aviation industry, the Center for Aerospace Research (CfAR) at the University of Victoria has been working on a Smart Auto-Flight Control System (SAFCS). After completion, the SAFCS is expected to continuously plan, execute and monitor airborne missions in which it is involved. It relies on System-Wide Information Management (SWIM) data protocols to interact with information providers and it uses the NASA WorldWind application as the virtual globe on which it operates.

There is a wide variety of algorithms that can be applied to solve path planning problems and, based on their most prominent features, they can be grouped in different classes. One class of interest for the overall project consists of sampling based algorithms, where samples are taken from a continuous space and computed plans are not restricted to discretized positions. Within this class, the Rapidly-exploring Random Tree (RRT) family deserves special attention due to its widespread applications and continuous development over the past years.

The present work aims to integrate the RRT planning family into the existing SAFCS application. More precisely, it is focused on identifying the algorithms that bring the most significant contributions, evaluating their adaptation to a 4D planning scenario and investigating if enhancements can be developed to improve their performances. In addition to these goals, the tuning parameters that mostly affect each algorithm are also investigated and performance analysis are conducted to compare the RRT with other planning families.

This report stars by providing the background needed to understand path planning algorithms in

general and then explaining the principle behind RRTs. Thereafter, implementation details concerning the integration into the SAFCS application are presented along with enhancements to existing algorithms that solve specific problems. Later, results are evaluated to analyze the effect of tuning parameters and compare performances between the RRT and other planning families. Finally, conclusions and suggestions for future work are given as a way to summarize what has been developed.

2. Background

The present work is focused on the sub-field of Planning and Decision Making, one of the core branches of both Artificial Intelligence (AI) and Robotics. First of all, it is important to clarify what is meant by the concept of planning. In this context, planning may be defined as the conversion of high-level specifications received from a user into a low-level set of actions that accomplish the specified objectives. A plan can, therefore, enclose the most variate sets of actions. Path planning, where a path is obtained as a result of the plan, is of special interest for this work.

2.1. Path Planning

A plan is a sequence of states, computed by a rational agent, that minimizes the cost needed to achieve a given goal in a particular environment. Regarding path planning, states correspond to locations of an agent and the produced plan is a path with a sequence of locations where the agent will pass. The most interesting cases for airborne systems are those considering 3D spaces. If the plan makes any time considerations (4D), then the path of positions becomes a trajectory of waypoints.

Being a topic of great interest over the past years, there is already some bibliography surveying the major methods currently implemented for path planning. However, in most cases, focus is on 2D scenarios and performance comparison between different algorithms is uncommon. For this reason, the work by Yang et al. [11] is of particular interest.

There is a wide variety of algorithms for 3D path planning and, dependent on the basic principle of each algorithm, it is possible to find similarities among some and aggregate them into general classes. Following the taxonomy proposed by Yang et al. [11], 3D path planning algorithms can be divided into 5 categories as shown in Figure 1.

From the categories presented, there are two of special interest because they contain the algorithms available in the SAFCS application. These are: node based optimal and sampling based algorithms. The former convert the problem of finding a path into a search problem and work with a predetermined set of nodes that represent the environment. They apply search algorithms that guarantee optimality and completeness within this particular representation. Their most significant planning family is the A* to which a detailed guide was developed by Ferguson et al. [3]. The latter require a general representation of the whole workspace and act by continuously sampling nodes from a continuous space within its frontiers. Although such methods are not restricted to a predetermined set of nodes, their optimality and completeness are, at best, only probabilistic.

2.2. Sampling Based Planning

Sampling Based Planning (SBP) has emerged to better cope with high-dimensional state-spaces and to avoid the initial discretization of the environment required by node based planning methods. It has proven to be an efficient approach to solve complex path planning problems with a reduced dependency on elaborate environment models. Elbanhawi and Simic [1] offer a detailed overview of what an SBP algorithm includes, discuss which functions and parameters mostly affect its performance and, finally, compare the results of various algorithms in different scenarios.

Yang et al. [11] further divide this class into passive and active, as presented in Figure 2. Passive algorithms generate a roadmap connecting the start and goal but require a search algorithm to operate in the generated network to find the best feasible path. On the other hand, active algorithms sample the environment and plan the path simultaneously and are, therefore, able to find feasible paths on their own. Some of the most implemented families include passive algorithms like the Probabilistic Roadmap (PRM), introduced by Kavraki et al. [7], as well as active algorithms like the Rapidly-exploring Random Tree (RRT), presented by LaValle [9]. While the RRT is the main focus of this work and will be discussed ahead, the PRM has also been integrated in the SAFCS and will be

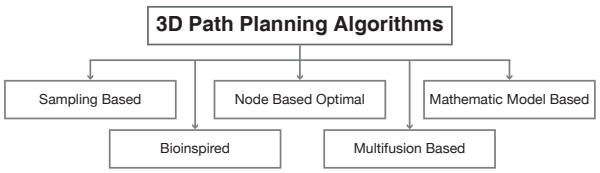


Figure 1: 3D Path Planning Taxonomy [11].

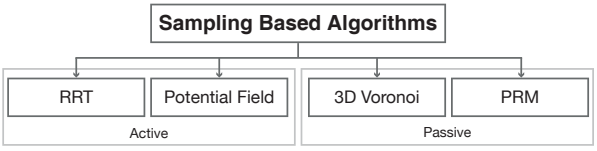


Figure 2: Sampling Based Algorithms [11].

used to compare performances.

It is important to state that SBP algorithms do not guarantee optimality or completeness. Most algorithms are probabilistic complete, meaning that, for a sufficiently large number of samples, they will eventually find a solution if there is one. Optimal solutions are hard to find in a continuous environment and the main goal for such planners is generally concerned with feasibility. Therefore, most SBP do not offer optimal solutions.

2.3. Rapidly-exploring Random Tree

The concept of an RRT was first introduced by LaValle [9] with the intention of exploring the benefits of already developed sampling-based planning techniques as well as improving the capability to handle nonholonomic systems and high degrees of freedom. Among the most advantageous properties of RRTs are its strong bias to expand towards unexplored regions, its probabilistic completeness under general conditions and the fact that it always remains connected even though the number of edges is small. The success of the RRT, notable since its first appearance, lead to more publications from Kuffner and LaValle [8] to better describe their work.

The RRT was a revolutionary algorithm but its initial version still lacked many of the desired characteristics for a high standard planner. Since the publication of the first article, two decades ago, several extensions have been proposed. Figure 3 presents some of the most prominent algorithms in RRT planning family.

One of the main disadvantages of the initial algorithm is its disregard for the cost associated with each path. For this reason, Urmson and Simmons [10] proposed a heuristic extension able to produce lower cost paths and deal with cost regions. Some years later, dynamic and anytime characteristics where first added to the basic planner and later fused together by Ferguson and Stentz [2] in an anytime dynamic RRT. More recently, the problem of optimality was addressed and Karaman et al. [6] presented a new algorithm entitled RRT* which marked the recent history of the RRT family significantly. Development of this planning family is still current focus of research and many other novel extensions have been proposed. However, they will not be discussed in the present work.

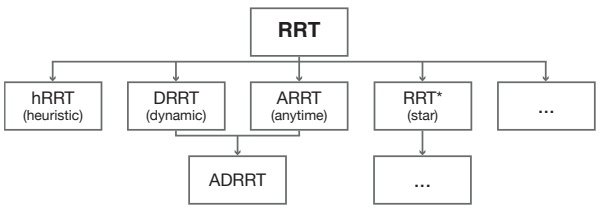


Figure 3: Existing algorithms in the RRT family.

Algorithm 1 Basic RRT. Adapted from [8]

1: **procedure** GROWRRT(q_{init}, q_{goal})
2: T.INIT(q_{init})
3: **for** $k \leftarrow 1, K$ **do**
4: $q \leftarrow$ SAMPLECONFIGURATION(`T.bias`)
5: $q_{near} \leftarrow$ NEARESTNEIGHBOR(q, T)
6: **if** EXTEND(T, q, q_{near}, q_{new}) $\neq$ *Trapped* **then**
7: **if** CHECKGOAL(q_{new}, q_{goal}) **then**
8: **return** COMPUTEPATH(T)
9: **return** *Failure*

10: **procedure** EXTEND(T, q, q_{near}, q_{new})
11: **if** NEWCONFIG(q, q_{near}, q_{new}) **then**
12: T.ADD_VERTEX(q_{new})
13: T.ADD_EDGE(q_{near}, q_{new})
14: CALCULATECOST(q_{new})
15: $T.C_max \leftarrow$MAX($q_{new}.cost, T.C\_max$)
16: **if** $q_{new} = q$ **then**
17: **return** *Reached*
18: **else**
19: **return** *Advanced*
20: **return** *Trapped*

Algorithm 1 (without Lines 14 and 15) shows an adaptation of the most basic RRT taken from the work by Kuffner and LaValle [8] which consists of two main procedures, the *GrowRRT* and the *Extend*. The first is responsible for growing a tree from the start configuration until the goal or a maximum number of iterations is reached. For every iteration, a random configuration is sampled and the nearest neighboring configuration already in the tree is computed. To better guide the search, it is usual that the sampling function occasionally returns the goal position with a frequency defined by a goal bias value, `T.bias`. These two configurations are used to attempt an extension of the tree and, if it succeeds, the resulting new configuration is compared with the goal region to see if it was reached. The second procedure implements the process which extends the tree in the direction of the sampled configurations. It first attempts to compute a path and get a new configuration, grown towards the sampled one from the nearest for a certain incremental distance ϵ. If it succeeds, a new vertex and a new edge are added to the tree and the status flag of *Reached* or *Advanced* is returned depending if the new configuration coincides with the sampled one or not. If no feasible configuration was computed, the status flag *Trapped* is returned.

There are some subtleties regarding the RRT, such as the metric to define the nearest neighbor and the expansion strategy to get a new configuration, but the algorithm is not complex. The tree avoids untraversable obstacles because only colli-

Algorithm 2 Heuristic RRT. Adapted from [10]

```
1:  procedure QUALITY(q)
2:      return 1 − (q.cost − T.C_opt)/(T.C_max − T.C_opt)

3:  procedure SELECTNODE(T)
4:      do
5:          q ← SAMPLECONFIGURATION(T.bias)
6:          q_near ← NEARESTNEIGHBOR(q, T)
7:          m_quality ← QUALITY(q_near)
8:          m_quality ← MIN(m_quality, T.floor)
9:          r ← RANDOMVALUE( )
10:     while r > m_quality
11:     return q, q_near

12: procedure SELECTNODEIK(T)
13:     while true do
14:         q ← SAMPLECONFIGURATION(T.bias)
15:         K_near ← NEARNEIGHBORS(q, T, k)
16:         K_near ← SORTBYQUALITY(K_near)
17:         for all q_near ← K_near do
18:             m_quality ← QUALITY(q_near)
19:             m_quality ← MIN(m_quality, T.floor)
20:             r ← RANDOMVALUE( )
21:             if r > m_quality then
22:                 return q, q_near

23: procedure SELECTNODEBK(T)
24:     do
25:         q ← SAMPLECONFIGURATION(T.bias)
26:         K_near ← NEARNEIGHBORS(q, T, k)
27:         q_near ← BESTQUALITY(K_near)
28:         m_quality ← QUALITY(q_near)
29:         m_quality ← MIN(m_quality, T.floor)
30:         r ← RANDOMVALUE( )
31:     while r > m_quality
32:     return q, q_near
```

sion free configurations are added, however, since no considerations are made regarding the cost of each configuration, the RRT does not take into account higher cost regions.

2.4. Heuristic RRT

Algorithm 2 shows the pseudocode of the procedures, presented by Urmson and Simmons [10], needed to extend the RRT to the heuristic RRT (HRRT) and guide the growth of the tree towards more promising regions. It presents a function to calculate the quality of a node as well as three different implementations of the *SelectNode* function, responsible for selecting the node for expansion given a random configuration. The first, selective sampling, is an iterative process that computes the nearest neighbor, as in the basic RRT, but only accepts sampled configurations whose near node has a sufficiently good quality. The second and third both choose the parent node from a set of k-nearest nodes from q sorted by quality. The difference between them is that the second procedure, iterative k-nearest, iteratively tests the quality of each possible parent while the third, best of k-nearest, only tests the quality of the best of the k-nearest.

To implement the heuristic RRT, the basic RRT algorithm (Algorithm 1) must be slightly modified. Lines 14 and 15 must be introduced so that the cost of a new node is set and the maximum cost value of nodes in the tree is updated. Lines 4 and 5 must be replaced by a call to one of the *SelectNode* functions, which returns both q and q_{near}.

The quality of a node is a relative value inversely proportional to its total cost, given by

$$m_{quality} = 1 - \frac{C_{vertex} - C_{opt}}{C_{max} - C_{opt}} \,. \qquad (1)$$

It is obtained from the difference between the total cost of a node, C_{vertex}, given by the sum of the cost from start and the heuristic to goal, and the cost of the optimal solution, C_{opt}, given by the heuristic between start and goal. It is then normalized by the difference between the maximum total cost of all nodes in the tree, C_{max}, and C_{opt}. The resulting value is then subtracted from 1 so that the quality of a node is a real number between 0 and 1, where 0 is the quality of the most expensive node in the tree and 1 is the quality of any node in the optimal path between start and goal.

Nodes with higher quality are more probable to be accepted for expansion but this probability may be limited depending on the probability floor value. This value is compared against the quality of a node and the minimum of the two is taken as the new quality. It is very important since it defines the maximum ratio of exploitation to exploration. In all *SelectNode* functions, nodes are selected for extension by checking its quality against a random value, $r \in [0, 1]$. If it is higher than the random value, the node is selected, otherwise it is discarded and a new node random configuration is taken. When the floor value is set to 1, the quality test will always be performed using the effective quality of each node. However, if the floor is set to a value close to 0, the quality test will most likely be performed using this value and all nodes have the same, very low, probability of being chosen.

3. Implementation

The main purpose of this work is to contribute to the development of the SAFCS application, which had already been started and whose general architecture was published by Heinemann et al. [5]. The application, henceforth referred to as the WorldWind Planner, extends the NASA World-

Wind SDK, a powerful Java development platform available for virtual globe applications. This open-source platform includes several interesting tools such as complex geometry computation and airspace representation, among several other things. Comprehension of the WorldWind Planner is required to fully understand the choices made to integrate the new features into the existing architecture.

3.1. WorldWind Planner

The WorldWind Planner is a Java application divided into several packages. Although code had to be refactored throughout the application to include new functionalities, the largest contributions were made to the "planning" and "ai" packages. While the planning package is where the general data structures required by the planners are defined, the ai package contains the algorithms that compute a trajectory for a certain scenario.

SWIM is one of the major motivations for the development of the SAFCS. In the WorldWind Planner, SWIM data is represented as planning obstacles that have a defined validity in both space and time. These obstacles require a properly indexed data structure and Heinemann et al. [4] propose using Cost Interval Trees associated with discretized regions of space.

Nonetheless, the core classes in the WorldWind Planner are, as the name suggests, the planners. Different planner classes realize different path planning algorithms and this work has extended the application to include algorithms of the RRT family. Planner classes are designed to follow object-oriented design principles and patterns, including the use of interfaces, to maximize re-use of existing functionalities. For this reason, a high level interface, the *Planner*, defines the methods which are required to implement any planner properly and are, therefore, common to all different planners. An abstract class, the *AbstractPlanner*, is also present at high level. It implements some methods which are not planner specific and includes certain variables required by all planners. All the implemented planners must then extend this abstract class or, if they are an extension of an already implemented planner, extend a class that extends this one. The most common structure for the various planning families is that the first implementation, like the basic RRT, extends the abstract class and the following, like the heuristic RRT, extend the first implementation. Figure 4 represents the described architecture.

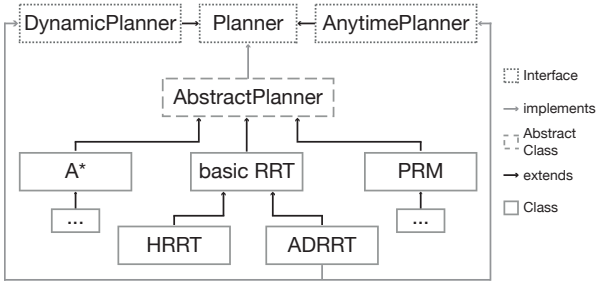


Figure 4: Planner Architecture.

3.2. Enhanced Heuristic RRT

Integration of the RRT into the WorldWind Planner was achieved without any problem and the second algorithm to be implemented was the HRRT. Introducing this new behavior as an extension of the basic RRT required few new methods. However, while evaluating the obtained results, two problems in the original HRRT algorithm were identified: it was incapable of dealing with scenarios where regions with abnormally high cost were present and the tuning of the exploitation ratio through the probability floor value was ineffective. Having identified these problems, enhancements to address both situations have been introduced to achieve the desired behavior the original HRRT was lacking.

The first and most severe problem is related to the formula for the quality of a node, which consists of a linear function given by equation (1). The use of a linear expression is adequate in cases where the costs of nodes in the tree are rather uniformly distributed between a minimum, C_{opt}, and a maximum, C_{max}. This is not the case if costs are not evenly distributed and there is an outlier node with a very high cost. In the limit, when the maximum cost of a node in the tree tends to infinity, all other nodes with non-infinite costs will have maximum quality regardless of their cost.

An illustrative example may be taken from a scenario where a direct path between start and goal is unobstructed but involves climbing a small hill, represented as a high cost region, which can be avoided by a path around it. If this is all that is present in the scenario, the original quality function is adequate and nodes inside the hill will have lower quality than those outside. This means that the HRRT algorithm will probabilistically only produce paths that go around the hill. However, if the scenario also includes, at any location, an enormous mountain, with a cost some orders of magnitude higher than the one of the hill, as soon as one node is extended inside the mountain region all others will be given the maximum quality. In this situation, the HRRT algorithm will avoid the mountain region but will disregard the small hill and the computed trajectories may or may not cross it.

Given this motivation, an exponential formula, more appropriate to compute the quality of a node, is proposed as

$$m_{quality} = 1 - \exp\left(-\frac{C_{opt}}{C_{vertex} - C_{opt}}\right) . \qquad (2)$$

This expression is of the type $1-\exp(-1/x)$ which, for the cases of interest where $x \geq 0$, leads to qualities in the range $]0, 1]$. The value for x is the relative difference between the total cost of a node, C_{vertex} and the cost of the optimal solution, C_{opt}. Since the quality is still inversely proportional to the cost and its range of values is the same, it can be directly replaced in the original HRRT algorithm. The maximum cost value of nodes in the tree is not included in the expression. Therefore, outliers with abnormally high costs do not affect the quality function and both scenarios previously described would lead to an identical result, avoiding the small hill. Removing the maximum cost from the formula has as its only downside a reduction in the robustness to over-simplified heuristic functions. The proposed formula is expected to be less efficient in situations where the estimated optimal cost is unrealistically less costly than the real optimal cost, i.e., different order of magnitude. However, heuristically guided algorithms presume the use of well chosen consistent heuristics, so this problem should never arise.

The second identified problem is related to the implementation of the probability floor value. As described by Urmson and Simmons [10], this value is supposed to tune the algorithms in favor of more exploitation of promising regions or exploration of the entire space. In the limit, if the value is set to full exploration, the algorithm should behave like the basic RRT.

In the original HRRT, this value was implemented in all *SelectNode* functions before the quality of the node was checked against the random value. The quality was compared with the probability floor and the minimum between them was tested against the random number,

$$m_{quality} = \min(m_{quality}, \text{prob\_floor}) \ . \qquad (3)$$

If the probability floor is set to 1, it is ignored by the minimum function. In this case, no value is filtered and the probability of a node to be chosen is equal to its quality. Therefore, nodes with higher quality are more prone to be chosen and the search is biased towards exploitation of promising regions. If the floor value is set closer to 0, the minimum function attributes such value to the majority of the nodes and their probabilities are identical but close to null. This means that most of the computation will be discarded due to not exceeding the random threshold value and the resulting computation time will be long no matter the scenario. When the floor is set to 0, the select node method enters an infinite loop since no random number will ever be lower than 0.

A different implementation is proposed where the maximum function replaces the minimum,

$$m_{quality} = \max(m_{quality}, \text{prob\_floor}) \ , \qquad (4)$$

Algorithm 3 Quality function Enhanced HRRT

```
1: procedure QUALITY(q)
2:     m_quality ← 1 − exp(−T.C_opt/(q.cost − T.C_opt))
3:     return MAX(m_quality, T.floor)
```

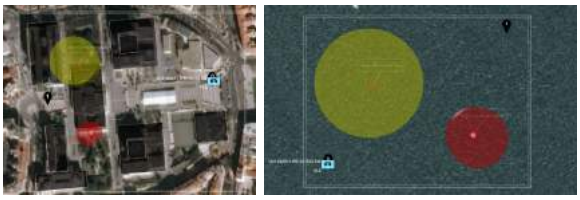
and, as a result, the probability floor has a reversed meaning. If the floor value is set to 0, it is ignored by the maximum function and the probability of a node to be chosen is coincident with its quality. This scenario corresponds to biasing the growth of the tree towards exploitation of promising regions and is equivalent to what was obtained by setting the floor value to 1 in the original HRRT implementation. For values closer to 1, the maximum function attributes this value to the majority of the nodes and they have an equally high chance of being selected, thus, promoting exploration of the entire space, behaving like the basic RRT, and discarding very few computations. This implementation not only has a better performance, but it also produces a more meaningful approach in which both the limit values for the probability floor, $p \in [0, 1]$, can be used and have well-defined meanings. Moreover, the implementation of the maximum function was also moved inside the quality method because this function is not only called directly by *Quality()* it is also called implicitly by the *SortByQuality()* and *BestQuality()* procedures for the iterative and best k-neighbors variants. If the probability floor is not considered inside the quality function, it does not have the desired effect because these sorting functions would use the quality of the node unfiltered by the probability floor.

To implement all discussed modifications, Lines 8, 19 and 29 should be removed from Algorithm 2 and the quality function should be replaced by the one in Algorithm 3. From here onwards, this new algorithm will be referred as the enhanced heuristic RRT (eHRRT).

4. Results

Once all algorithms have been implemented, it is important to analyze the obtained results to confirm theoretical assumptions and to evaluate performances. Since all RRT planners have a stochastic behavior, they can only be analyzed statistically. For this reason, all the presented data was taken from multiple runs for each algorithm with a particular set of parameters in a certain scenario.

The scenario in which a planner is tested deeply affects the results it provides and it is important to describe in which conditions the following results have been obtained. For the evaluation of the developed planners, two scenarios have been used and are depicted in Figure 5. Black boxes represent terrain



(a) Scenario I (b) Scenario II

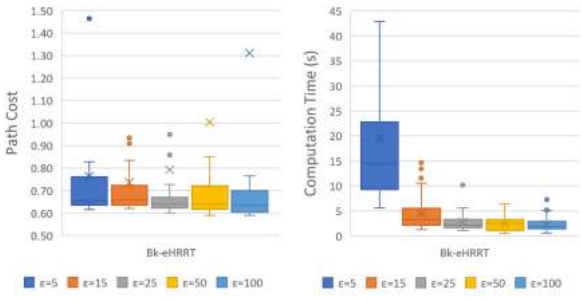
Figure 5: Experimental Scenarios.

and colored cylinders are SWIM obstacles, where red is a more costly region than yellow. The white box delimits the environment and waypoints 0 and 1 mark the start and goal positions, respectively. While Scenario I is rather difficult for planning because it has many obstacles and narrow passages, Scenario II is simpler because it has more free space and does not contain terrain obstacles. In both scenarios, the ceiling was set so that no path can be computed over the obstacles. Moreover, the optimal path cost should be inferior to 1 because distances considered are normalized by the spatial diagonal of the box and the best obstacle free trajectory has a length inferior to one diagonal.

4.1. Parameter Tuning

In general, algorithms have a typical behavior but can be fine tuned by adapting their key parameters to optimal values. There are several tuning parameters in the RRT family, but the incremental distance, ϵ, is the most important because it affects the behavior of every algorithm. It represents the maximum distance allowed for each extension and tuning this value represents a trade-off between resolution and computation time.

This parameter should take both the environment and aircraft into consideration. To ensure that the entire space is covered, ϵ values should not be too small when compared with the longest dimension of the bounding box. Moreover, this incremental distance also implies a temporal resolution related to the aircraft speed. Depending on the mission, it is common to have a defined order of magnitude for the time to go between waypoints. This value can be multiplied by the aircraft cruise speed to obtain an estimate for the distance associated.



(a) Path Cost (b) Computation Time

Figure 6: Incremental Distance Tuning.

This parameter was tuned for a commercial quadcopter, with a cruise speed of 15m/s, flying in Scenario I, whose diagonal has a length of 500m. Considering these values, the enhanced HRRT (best-k) has been tested with a set of incremental distances, $\epsilon = \{5, 15, 25, 50, 100\}$m, and results are presented in Figure 6.

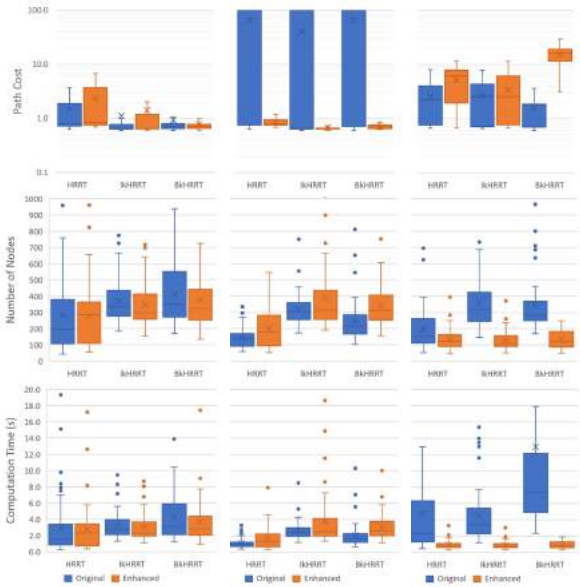
The first impression taken from the graphics is that the computation time decreases as ϵ increases and larger edges are allowed. Nonetheless, it is important to notice that little gain is achieved by increasing ϵ to values higher than 25m, meaning that the gain has a limit. Regarding the path cost, the average (crosses in the graph) for the HRRT increases with ϵ due to the presence of more outlier cases where the final path had a very high cost (dots in the graph). Decreasing the value from 15m to 5m produces worse results, meaning there is also a limit to the path cost gains for low values. For these reasons, $\epsilon = \{15, 25\}$m are considered the two most appropriate values for this aircraft in this scenario.

4.2. Enhanced HRRT

To evaluate the effectiveness of the proposed enhancements to the heuristic RRT, performances of the original (HRRT) and enhanced (eHRRT) algorithms were compared. To promote the existence of abnormally high costs, the Risk Policies defined for the SAFCS were used to establish the maximum local cost admitted for each mission. If an obstacle has a cost higher than the one allowed by the policy, all waypoints that lie inside it are given an infinite cost. Considering Figure 5, for a risk policy of *Safety*, the yellow obstacle respects the risk policy and is given a finite cost while the red does not and has infinite cost. If a risk policy of *Ignorance* is used, both obstacles have finite costs.

To achieve significant results, the planners were run with three different combinations of risk policies and probability floors. First, the risk policy was set to *Ignorance*, so that both obstacles had a finite cost, and the probability floor to promote exploitation ($p_{HRRT} = 1$ and $p_{eHRRT} = 0$). Then, the risk policy was changed to *Safety*, meaning the red cylinder received an infinite cost, and the floor value was kept equal to promote exploitation. Finally, the risk policy was set back to *Ignorance* and the floor value was changed to promote exploration ($p_{HRRT} = 0.1$ and $p_{eHRRT} = 1$). During all tests, the incremental distance was kept constant ($\epsilon = 15$m), as well as the goal bias ($b = 5\%$) and the number of k-neighbors to consider ($k = 5$).

Figure 7 shows the performance of the HRRT al-



(a) Ignorance (b) Safety (c) Exploration

Figure 7: Heuristic Planner Comparison.

gorithm (h-, Ik- and BkHRRT) in the different scenarios for both original and enhanced versions. The obtained data is relative to the cost of the computed path, the number of nodes added to the tree and the computation time until a solution was reached. For the purpose of representation, paths with an infinite cost are set to 100.

Figures 7(a) shows that, for a situation without very high costs and where exploitation is desired, both HRRT and eHRRT produce similar results. As expected, it can also be noted that the hRRT variation creates fewer nodes and produces more costly solutions than the more aggressively exploiting BkHRRT. Figure 7(b), is the one that best describes the deficiency of the original HRRT to deal with abnormally high cost regions. For this scenario, the three variations of the HRRT all produce solutions with infinite cost while the eHRRT produces trajectories with costs lower than 1. The number of nodes created and the computation time is lower for the HRRT because the search is no longer well guided and all configurations have a high probability of being accepted.

To give a more illustrative representation of what is happening in this scenario, Figure 8 depicts the trajectories produced by the original and enhanced HRRT for a scenario of exploitation with a risk policy of *Safety*. As discussed, the presence of a very high cost region invalidates the exploiting properties of the original HRRT and it finds a random path to goal crossing the high cost region in yellow. Meanwhile, the eHRRT is not affected by the infinite cost region and finds a low cost trajectory to

(a) Original BkHRRT (b) Enhanced BkHRRT

Figure 8: Final Trajectory (Exploiting in Safety).

the goal.

When the probability floor is set to promote exploration, the algorithms should behave like the basic RRT and produce costly solutions in a short computation time and with few nodes created. From Figure 7(c), it can be seen that only the enhanced implementation follows this behavior since the original HRRT produces lower cost solutions and expands considerably more nodes. Moreover, the computation times in this scenario prove the actual effect of the probability floor value in the original HRRT. To produce paths with a similar cost and number of created nodes as the scenario of exploitation in *Ignorance*, all the original HRRT variations require high computation times. Therefore, it can be concluded that most of the sampled nodes are discarded since, for a similar number of nodes in the final tree, the computation time is much higher.

4.3. Performance Analysis

The various RRT planners have been implemented because they have different characteristics. For the considered scenarios, several of these planners could be used and the computed trajectories would vary. For this reason, it is interesting to compare the results obtained for each algorithm. From the ones implemented, the RRT, HRRT and ARRT were selected to be evaluated in scenario I, with a risk policy of *Ignorance*, and Figure 9 shows the data collected. To compare performances, the same parameters were kept constant with the same values as in the previous section.

It is possible to conclude that, having outperformed the original HRRT, the eHRRT computes solutions comparable to the more complex ARRT and both consistently provide very low cost solutions. However, the ARRT runs for approximately 10 times longer. Even though the eHRRT does not have anytime capabilities and takes longer to achieve a feasible path, in many cases it is the most efficient algorithm.

To better illustrate the trajectories produced by each algorithm, Figure 10 depicts the final trajectories of the RRT and ARRT. It can be seen that the RRT disregards path cost and has an uneven trajectory crossing a high cost region. On the other

hand, the last iteration of the ARRT produces a low cost solution which is similar to what is obtained by the eHRRT in Figure 8(b).

4.4. Other Planning Families

Since the WorldWind Planner also includes the PRM and A* families, it is interesting to compare the RRT (Bk-eHRRT) with these families working in the same environment. The algorithms have been tested with a set of 100 different start and goal positions and every planner was run once for a pair start-goal. Because the optimal path cost varies depending on the location of both start and goal, the path cost presented was normalized by the euclidean distance between the two positions. For unobstructed paths, the optimal cost should be 1.

Taking the values produced by the RRT as a reference to normalize the cost and computation times of the other planners, a performance measurement is given by

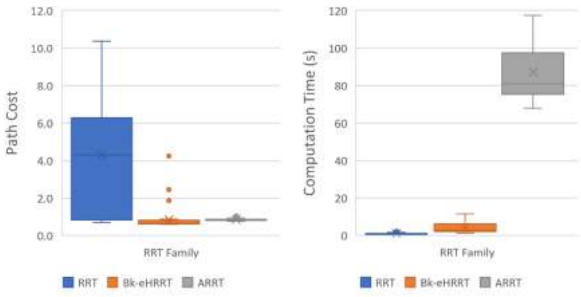
$$\eta_x = \frac{1}{c_{rel_x}} \cdot \frac{1}{t_{rel_x}} , \qquad (5)$$

where c_{rel_x} and t_{rel_x} are the cost and time of algorithm x normalized by the ones for the RRT.

Performances were compared in scenario II and the results obtained are shown in Figure 11. Both graphs show the particular value for each iteration as well as a box plot with the statistical data after 100 iterations. Lines in 11(a) represent median values after all iterations and lines in 11(b) show the average computation time after each iteration.

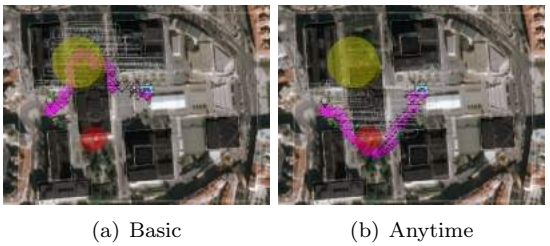
Comparing the costs of the RRT and the PRM, both present low dispersion values and their medians are very similar. However, the existence of more outliers elevates the average value of the RRT when compared to the PRM. The median value for the A* is slightly higher compared to both other algorithms. Moreover, it never achieves the unitary cost associated with a straight line path because it is limited to the waypoints of its planning grid and is not capable of planning in the space in between.

Regarding computation time, an interesting measure is the average after each iteration. Since the



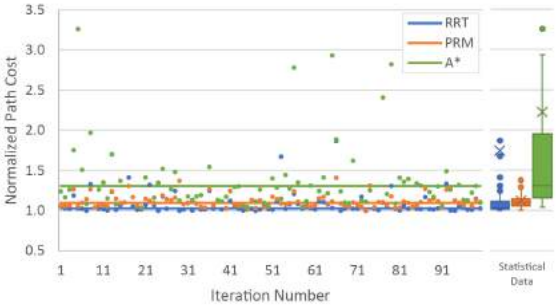
(a) Path Cost (b) Computation Time

Figure 9: RRT Family Comparison.

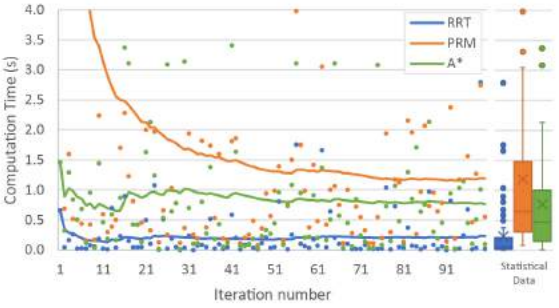


(a) Basic (b) Anytime

Figure 10: Final Trajectory.



(a) Normalized Path Cost



(b) Computation Time

Figure 11: RRT PRM A* Comparison.

PRM takes a long time to construct its roadmap, its first iteration has a very high computation time. However, the constructed roadmap is reused for the next iterations and the computation time is diluted as the number of iterations increases. The RRT does not require any extra time for the first iteration because it constructs a tree for every case. It can be seen that the average computation time of the PRM decreases with iterations. However, in the given scenario, growing a tree is very fast and the PRM never becomes quicker than the RRT. Similarly to the RRT, the computation time of the A* is independent of the iteration and varies only depending on the complexity of the planning case. The RRT is clearly the fastest algorithm in this scenario and the A* is slightly faster than the PRM.

Table 1 presents the values obtained for this scenario. Even though the final performance measurement rates the RRT considerably higher than the

Alg.	Cost		Time		η
	Median	Rel.	Median	Rel.	
RRT	1.03	1.00	0.21	1.00	1.00
PRM	1.10	1.07	1.33	6.49	0.14
A*	1.31	1.27	0.82	4.01	0.20

Table 1: Planner Comparison for Scenario II.

others, comparing algorithms with different characteristics is not trivial and this performance measurement may overestimate the importance of computation time. Nonetheless, results show that the RRT has clear advantages in this scenario.

5. Conclusions

The major achievement of the presented work was the successful adaptation of existing planning algorithms of the RRT family to the WorldWind Planner application. As a direct result of this work, RRT planners are now available and possess all the most relevant characteristics required by the SAFCS project.

From the numerous RRT algorithms, the HRRT, DRRT, ARRT, ADRRT and RRT* were chosen as the ones that best fit the requirements of the SAFCS project. They all proved to be adaptable to the 4D scenario at hand by sampling configurations in a 3D space and then propagating time between waypoints considering all the constraints. In some of these algorithms, performance issues were identified and solutions were proposed. These solutions were especially important for the HRRT planner, where fundamental problems were found, and improvements led to a significant extension of the algorithm, the enhanced HRRT (eHRRT).

After implementation, it was possible to evaluate how each tuning parameter affected the results obtained and compare the performance of the RRT with the PRM and A* families. Regarding parameter tuning, no optimal values were found but relations between parameters and results were identified and it is now possible to qualitatively set values for the desired results. Regarding the comparison between different planning families, it was not easy to compare algorithms with so many differences and to quantify their relative performances. For this reason, it is not possible to define the best planning family, but the RRT clearly proved its advantages.

Regarding future work, all implemented algorithms should be further evaluated. Certain algorithms could be improved if backwards planning was achieved. However, this is non-trivial in 4D when there is no estimated time of arrival. A possible solution could be related with the use of time slots. Other possible improvements could come from defining the sampling environment as hierarchical. This would allow certain regions of interest to be refined to obtain better results. However, the RRT samples and plans simultaneously and the implications of a hierarchical environment should be further investigated.

References

[1] Mohamed Elbanhawi and Milan Simic. Sampling-based robot motion planning: A review. *IEEE access*, 2014.

[2] Dave Ferguson and Anthony Stentz. Anytime, dynamic planning in high-dimensional search spaces. In *Proceedings 2007 IEEE International Conference on Robotics and Automation (ICRA'07)*, 2007.

[3] Dave Ferguson, Maxim Likhachev, and Anthony Stentz. A guide to heuristic-based path planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS'05)*, 2005.

[4] Stephan Heinemann, Hausi A Müller, and Afzal Suleman. UAV 4D grid-based trajectory planning using SWIM data and interval trees. Unpublished Draft: `github.com/stephanheinemann/worldwind`.

[5] Stephan Heinemann, Hausi A Müller, and Afzal Suleman. Towards a smarter autoflight control system infrastructure. *Journal of Aerospace Information Systems*, 2018.

[6] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using RRT*. In *Proceedings 2011 IEEE International Conference on Robotics and Automation (ICRA'11)*, 2011.

[7] Lydia E Kavraki, Mihail N Kolountzakis, and J-C Latombe. Analysis of probabilistic roadmaps for path planning. In *Proceedings 1996 IEEE International Conference on Robotics and Automation (ICRA'96)*, 1996.

[8] James J Kuffner and Steven M LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of 2000 IEEE International Conference on Robotics and Automation (ICRA'00)*, 2000.

[9] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[10] Chris Urmson and Reid Simmons. Approaches for heuristically biasing RRT growth. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, 2003.

[11] Liang Yang, Juntong Qi, Dalei Song, Jizhong Xiao, Jianda Han, and Yong Xia. Survey of robot 3D path planning algorithms. *Journal of Control Science and Engineering*, 2016.