Probabilistic Roadmaps for Aircraft 4D Path Planning

Henrique Ferreira
henriquembferreira@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Portugal

April 2019

Abstract

The responsible entities for Air Traffic Management (ATM) around the world are integrating innovative procedures to cope with the constant rise in air traffic. New technologies such as System Wide Information Management (SWIM) aim at improving the efficiency and safety levels of operations while reducing costs and emissions. The SWIM program releases an overwhelming amount of aviation information in a machine-readable format, which is not fully exploitable by humans. To leverage aviation data, a Smarter Auto-Flight Control System (SAFCS) is being developed, which continuously monitors, analyzes, plans and executes airborne missions. It supports pilot decision making by suggesting routes computed by path planning algorithms. Probabilistic Roadmap (PRM) is a path planning algorithm particularly efficient at dealing with high dimensional problems and multiple queries. First, it creates a graph by sampling the planning environment and then resorts to a search algorithm for computing a path. Over the last two decades, PRM has been subject to several research papers, some of which extend it to include anytime, dynamic and online capabilities. The SAFCS framework and the integration of the PRM planning family are meticulously discussed. Solid reasoning is provided for the changes introduced relatively to the original algorithms. Lastly, the planners' performance is subject to evaluation and the conclusions drawn from results are stated.
Keywords: Automatic Flight Control, Path Planning, Sampling-based algorithms, Probabilistic Roadmaps, Anytime planning, Dynamic planning

1. Introduction

The European project Single European Sky ATM Research (SESAR) proposes an overhaul of the European airspace and its ATM, by introducing better technologies and procedures. Overall, it expects to improve the flight predictability, airport's and airspace's capacities, safety and to reduce costs, fuel consumption and emissions. Equivalently, FAA is carrying out the NextGen program with relatively the same goals.

The seek for more efficient technologies is being conducted on several areas. *System Wide Information Management* (SWIM) aims to enhance how the information is shared across all aeronautical participants. Essentially, it intends to deliver the appropriate information to the users at the right moment, and contribute for a better situational awareness (SA). *Controller-Pilot Data-Link Communications* (CPDLC) is an air-ground telecommunications link which allows the exchange of data between controllers and pilots. This includes a set of text messages, such as clearance, information and request procedures, that can replace or complement the traditional voice communication method. *Initial four-dimensional trajectory management* (i4D) improves the flight predictability and air traffic flow by making sure that trajectories (position and time) are always synchronized between air and ground stakeholders.

Nowadays, auto-flight control systems do not take into account all the obtainable data and the flight quality depends greatly on the pilot's performance. Stephan Heinemann proposes, in his PhD dissertation [5], the development of a Smart Auto-Flight Control System.

1.1. Smart Auto-Flight Control System

The Smart Auto-Flight Control System (SAFCS) aims at creating an infrastructure which continuously monitors, analyzes, plans and executes airbone missions. SAFCS is being conducted in Victoria, with the support of the University of Victoria (UVic), namely the Rigi Research and Centre for Aerospace Research (CfAR) labs.

The main goal behind SAFCS development is to investigate and create systems that enable safer and more efficient air transport solutions. The error-prone nature of human beings and its faulty sensory capacity are significant challenges for airborne operations. Instead, automatic systems are better in accumulating and processing the available data

in order to support pilot decision making.

Heinemann et al. [6] mention the required information to set up a suitable situational awareness for decision making. Aircraft data such as its capabilities and systems' status are indispensable, as well as the current flight phase (e.g., take-off, departure, cruise, arrival, approach or landing). Also, the physical context including air and ground environments are of utmost importance. Furthermore, the company goals, human factors, other operational aspects and the criticality of the situation (e.g., diversion or emergency) are essential for establishing an appropriate SA.

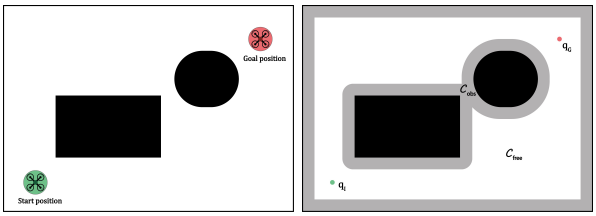
The Smarter Auto-Flight Control Systems infrastructure proposes a shift in the paradigm of aircraft systems and its development is based on the intertwinement of three different research fields: *Planning and Decision Making*, *Autonomous Flight and Guidance*, *Self-Adaptive and Self-Managing Systems*.

Planning and Decision Making is one of the most important branches of Artificial Intelligence. Planning consists of finding a chain of actions to be performed by an rational agent in order to accomplish a specific goal. The proposed smart auto-flight control system must compute and revise plans for carrying out airborne missions. More specifically SAFCS intends to compute flight paths for mobile agents (aircraft) from an initial to a goal condition optimizing applicable performance measures. For this reason, SAFCS can be seen as a Path Planning problem as commonly designated in Robotics.

2. Background

Path planning is a subset of planning problems particularly relevant in Robotics, Computer Science and Autonomous Vehicles. In order to solve a generic planning problem, it is crucial to define the state space. For a path planning problem it corresponds to the set of possible transformations that can be applied to a robot $\mathcal{A}$. This is commonly referred to as the configuration space $\mathcal{C}$, based on the work of Lozano-Perez [11], who first utilized this nomenclature in the context of planning. The $\mathcal{C}$-space is a key concept for path planning as it encompasses the set of all robot states or configurations. The workspace $\mathcal{W}$ of a path planning problem is frequently designated as the set of positions in which the robot body $\mathcal{A}$ works and can be either a 3D space or a 2D surface. In order to define the workspace, a world model, which can either be given to the robot offline or can be obtained by sensing, is fundamental.

A path planning algorithm or planner consists of the procedure to follow in order to successfully find a valid path between the start and goal configuration. LaValle [10] dissects planning algorithms



(a) Workspace representation (b) $\mathcal{C}$-space representation

Figure 1: Path planning problem

and their theoretical attributes. Yang et al. [13] review some of the established 3D path planning algorithms, analyzing their advantages and limitations.

Node based optimal algorithms produce a path by exploring a discretized set of configurations. The algorithm solely searches through the graph employing appropriate *heuristics* in order to avoid examining the whole state space. Algorithms such as A* [4] find the optimal path within the decomposed graph.

Probabilistic sampling based algorithms start off from a continuous environment and create a discrete tree or roadmap. This approach is non-deterministic since it uses probabilistic sampling to generate a set of nodes or configurations. Some algorithms of this kind are able to concurrently compute a path and sample new states (active), others just focus on building the graph and rely on node based optimal algorithms to search for a path (passive). Probabilistic Roadmap (PRM) is one of the most known algorithms of this family, and the main focus of this work.

Regardless of the type of the algorithm, some planner features are considered to be extremely important in the SAFCS scope. An *anytime planner* computes a sub-optimal collision-free path as fast as possible. Then, the algorithm incrementally improves the solution as long as it is running or according to a fixed deadline. A *dynamic planner* deals efficiently with changes in the environment and in the agent capabilities. It maintains the unaltered knowledge from previously computed solutions and discards only the faulty information.

3. Probabilistic Roadmap

Kavraki et al. [9] developed the original probabilistic roadmap method specially for robots with many degrees of freedom. The algorithm is divided into two phases: *Learning Phase* which focuses on building the roadmap ensuring that it maintains acceptable levels of connectivity even in narrow passages or around obstacles and *Query Phase* which finds a path between the initial and goal state. Several queries can be called on the same roadmap, with dif-

ferent conditions, making it a multiple query planner.

The *Learning Phase* is comprised of two sequential steps. The main goal of the first one - Construction step - is to have a uniform coverage of $\mathcal{C}_{free}$ by sampling the majority of the configurations of the final roadmap. Then, the Enhancement step improves the roadmap connectivity, mainly on more difficult zones.

The pseudocode for the Construction Step is outlined in Algorithm 1. It starts out with an empty graph $G = (N, E)$. Then, a new random free configuration q is sampled and added to the set of configurations N. After that, a selection of configurations of N is made, according to a certain strategy, in order to find candidate neighbors N_q (Line 7). For all configurations in N_q in order of increasing distance from q, a local planner is used to find if a connection between the two configurations is possible. If the two nodes are not in the same connected component of the graph and the local planner validates the connection $\Delta(q, q')$ (Line 9), a new edge is created and added to the graph (Line 10). This process is repeated until a maximum X number of configurations is generated or until a time limit is reached.

Algorithm 1 PRM - Construction Step

```
1:  procedure ConstructPRM
2:      N ← ∅
3:      E ← ∅
4:      loop X times
5:          q ← RandomNode()
6:          N ← N ∪ {q}
7:          N_q ← a set of candidate neighbors of q chosen from N
8:          for all q' ∈ N_q, in order of increasing D(q, q') do
9:              if ¬same_connected_component(q, q') AND △(q, q') then
10:                 E ← E ∪ {(q, q')}
11:                 update connected components
```

The enhancement step selects a finite number of nodes, which are considered to be difficult according to an established measure, and expands them. The expansion of a configuration n is realized by sampling a free configuration q from its vicinity. Then, the new configuration is added to the set N and connected to other nodes, in the same manner as in the Construction Step. Figure 2b represents an example of PRM after the Enhancement Step.

The roadmap constructed can be utilized for many different pairs of start and goal configurations. A query includes the specification of the initial configuration q_I and final configuration q_G and requires as an output a feasible path. First, the initial and final configurations are connected to some two nodes in the graph. Then, a path is computed along the edges of the graph, resorting to a node based optimal algorithm such as A* (Figure 2c).

Authors in [8] suggest a simplified version of the original PRM algorithm (sPRM) in order to facilitate the analysis of the algorithm's performance. This approach leaves out the Enhancement Step, and allows the connection of nodes in the same component. Karaman and Frazzoli [7] proposed an asymptotically optimal version of the PRM algorithm, with increased computational efficiency when compared to sPRM. PRM* adopts a neighbor selection technique based on variable distance or variable k-nearest. Bohlin and Kavraki [2] introduce a new variation of the PRM algorithm, proposing a new procedure for collision checking. In Lazy PRM, the sampled roadmap is initially constructed without collision considerations, therefore consisting of a set of paths assumed to be feasible. The feasibility of the solution is only evaluated after a path is found during the *Query Phase*.

3.1. Flexible Anytime Dynamic PRM

Authors in [1] introduced an anytime and dynamic version of the probabilistic roadmap planner capable of dealing with region preferences. The flexible anytime dynamic probabilistic roadmap (FADPRM) moves away from the original multi-query approach since the roadmap is built or extended considering the current goal.

FADPRM integrates the Generalized Adaptive A* (GAA*) [12] algorithm to, simultaneously, explore the configuration space and compute a feasible path. The quality of a path depends on its desirability value − *pathdd* − and its cost − *pathCost* − in terms of distance traveled by the robot. The expression for $g(s)$ is defined as follows:

$$g(s) = \frac{pathdd(s_{goal}, s)}{1 + \gamma \cdot pathCost(s_{goal}, s)} \tag{1}$$

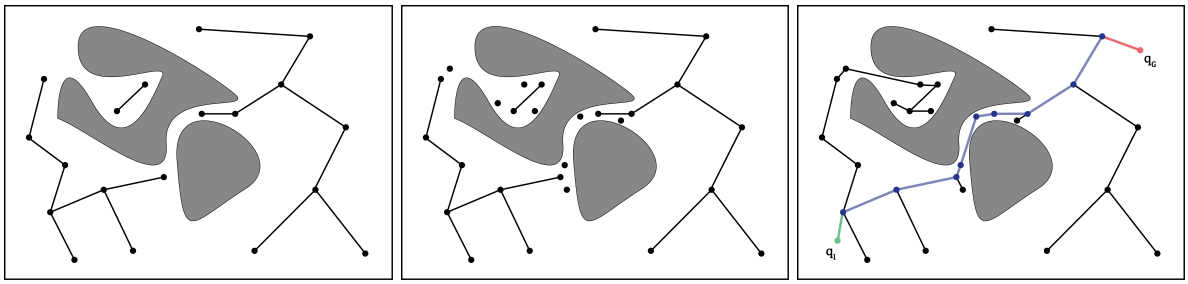
The parameter γ assumes values between 0 and 1 and adjusts the importance of dd on $g(s)$ and $h(s)$. The f-value is computed exactly as in A*, with an extra normalization factor of 2, thereby ensuring that its values range from 0 to 1.

$$f(s) = \frac{g(s) + h(s)}{2} \tag{2}$$

The selection of the node to expand follows a slightly different thought process than in A*. Nodes in *OPEN* are expanded in decreasing priority according to the key function $key(s)$, which is a pair $\left[k_1(s)\ ,\ k_2(s)\right]$ defined as follows:

$$key(s) = \left[\frac{1-\beta}{density(s)} + \beta \cdot f(s)\ ,\ h(s)\right] \tag{3}$$

The inflation factor β of the algorithm varies between 0 and 1. It is easily understandable that



(a) Roadmap after Construction Step (b) Roadmap after Enhancement Step (c) *Query Phase* of PRM

Figure 2: An example of a computed roadmap by the PRM algorithm

for β equal to 0, the selection of a node to expand does not take into account neither the *dd* nor the $pathcost(s)$ of the nodes. Oppositely, with $\beta = 1$ the considerations about the density of the node are left out and the node selected follows a best-first strategy just like A*. The MAIN procedure in FADPRM initializes this parameter to a low value, in order to find a suboptimal path as fast as possible. Then, if no changes in edge costs are detected, β is increased by a small quantity and a new path is computed again. With this β based implementation, the algorithm behaves in an anytime manner.

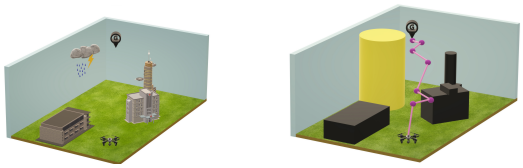
Dynamic changes in the environment affecting edge costs are dealt by a CONSISTENCY PROCEDURE adapted from GAA*, but with a similar *modus operandi*.

COMPUTEORIMPROVEPATH() is the core method of the FADPRM planner, where the path is actually computed. The process consists of continuously removing the node s with the maximum key present in *OPEN* and expanding it. In each iteration the h-values of the neighbors are properly updated, resorting to the UPDATESTATE function to revise the g and h-values of the nodes. The method finishes when the extracted node is connectable to the goal.

4. Implementation

Heinemann et al. [6] explain in detail the architecture of the SAFCS project and the implementations made prior to the initiation of this work. The core SAFCS[1] component was developed by extending the NASA Worldwind Software Development Kit (SDK). Worldwind is an API developed by NASA and implemented in Java. It provides a virtual globe and a variety of geometric objects along with visualization and intersection methods.

The SAFCS core component aims at solving an augmented 3D path planning problem. Given an initial position and time for a certain aircraft, the objective is to compute a path to achieve the goal position, while avoiding untraversable obstacles and



(a) Illustration of a real-world example (b) Representation in SAFCS

Figure 3: The SAFCS 4D path planning problem

bearing in mind SWIM data. Because of the temporal considerations inherent to movable obstacles and consequently to the planning process, SAFCS is considered to be a 4D path planning problem for aircraft. Figure 3(a) illustrates a real-world example of such a context. The corresponding representation in terms of the SAFCS framework is shown in Figure 3(b), with three untraversable obstacles in black, one SWIM obstacle in yellow and the path in pink.

The Probabilistic Roadmap planners are located within the `prm` sub-package which is a part of the `ai` package. All planning algorithms in the SAFCS architecture are necessarily subject to an `aircraft`, `environment`, `costPolicy` and `riskPolicy`. However, the singularities of each planner impel the creation of specific attributes and methods. As a probabilistic sampling planner, the whole PRM family is only supported by the `SamplingEnvironment`.

4.1. `Rigid PRM`

The original PRM was subject of several research papers and new versions were proposed with improvements on different steps of the algorithm. Nevertheless, it was realized that many of the new techniques suggested were independent from each other. Therefore, a single algorithm was developed – `RigidPRM`[2] – which integrates a handful of methods for all different aspects of the algorithm. A cen-

[1] `https://github.com/stephanheinemann/worldwind/tree/feature/continuum-planner`

[2] `https://github.com/stephanheinemann/worldwind/tree/feature/continuum-planner/src/main/java/com/cfar/swim/worldwind/ai/prm/rigidprm`

tralized approach was privileged in detriment of a horizontal or vertical architecture. The chosen approach integrates the knowledge from the original PRM, sPRM, Lazy PRM and PRM* in an unique algorithm. The UML layout of `RigidPRM` is presented in Figure 4, with a rather high number of user inputs that enable its customization.

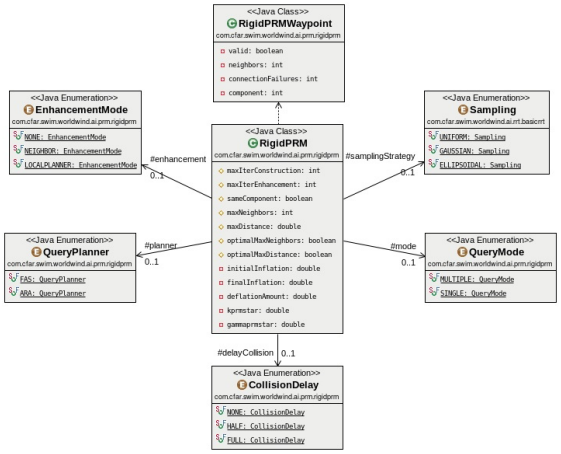


Figure 4: UML representation of `RigidPRM` implementation

The *Sampling parameters* include the specification for the number of waypoints to be sampled in the Construction step – `maxIterConstruction` – and in the Enhancement Step – `maxIterEnhancement`. Also, the `Sampling` strategy to be used in the Construction step is a selectable attribute as well as the `EnhancementMode` (the technique used to select which nodes should be expanded in the Enhancement step).

The *Connection parameters* controls the selection of neighbors. The `sameComponent` variable allows or prohibits connections between `waypoints` in the same connected component. Also, provides the k-nearest – `maxNeighbors` – and maximum distance – `maxDistance` – options as neighbor selection techniques. Finally, two booleans attributes represent the approaches designed in PRM* (`optimalMaxNeighbors` and `optimalMaxDistance`).

The *Query variables* are composed of the `QueryMode` and the `QueryPlanner`. The first one comes with two options: `SINGLE`, which creates a roadmap from scratch even if there is already one, and `MULTIPLE`, which maintains the previously sampled graph. The `QueryPlanner` is the search algorithm which finds the path from the start to the goal. Extraordinarily, there are some anytime parameters (`initialInflation`, `finalInflation`, `deflationAmount`) that are required if Anytime Repairing A* is selected as the `QueryPlanner`.

Algorithm 2 Rigid PRM

```
1:  procedure Main( )
2:      if QueryMode is SINGLE then
3:          N ← ∅
4:          E ← ∅
5:          ConstructionStep(Sampling, Connection, CollisionDelay)
6:          EnhancementStep(EnhancementMode, Connection, CollisionDelay)
7:      else if QueryMode is MULTIPLE then
8:          Maintain previously computed graph G = (N, E)
9:      Add W_origin and W_destination to the graph
10:     FindFeasiblePath(G, QueryPlanner, CollisionDelay)
```

4.2. Flexible Approach

The flexible approach implemented in SAFCS was highly inspired in the original FADPRM algorithm. Nonetheless, some changes were introduced in order to improve specific aspects of the original algorithm. The multiple query nature and reusability of PRM were vital characteristics that were carefully maintained, as it traditionally distinguishes PRM from other planning algorithms. A vertical and incremental architecture was chosen, as it illustrated in the UML diagram of Figure 5. First, a flexible and anytime version was developed (FAPRM). Then, it is extended to include dynamic planning capabilities (FADPRM), and lastly to have repairing abilities (RADPRM).

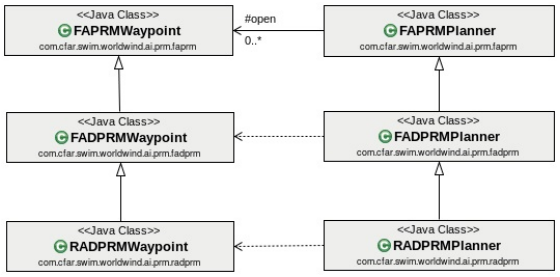


Figure 5: UML representation of the architecture of flexible PRM

One significant change is related to the computation of g, h and f-values as well as the integration of `DesirabilityZones` in the cost function. As a forward search algorithm, the g-value represents the cost or quality of the best path from the start to a given `waypoint`, while h is the estimated cost to the goal according to the heuristic function.

The original A* algorithm requires a non-decreasing nature for g (because distances are always non-negative). For the case of FADPRM, where *pathCost* appears in the denominator, it would be expected to have a non-increasing evolution. However, that cannot be ensured, since the old expressions depend on the degree of desirability of the path (*pathdd*). In fact, an increase in *pathdd*

might lead to a greater g-value. For this reason, the new expressions for g and h do not include explicitly a desirability variable. Instead, the desirability factor is included in the computation of the cost between two `waypoints`. The cost function for this specific algorithm incorporates a desirability multiplier Γ. It is computed based on the parameter γ of the algorithm and the desirability of the considered `edge` dd^e (Equation 4).

$$\Gamma = 2\gamma + \frac{1-\gamma}{dd^e} - 1 \tag{4}$$

Figure 6 pictures the logarithmic evolution of Γ based on admissible values for γ and `edge` desirability dd^e. Even though the face value of Γ is used during the cost step, the logarithm was used for representation purposes. As intended, for $\gamma = 1$ the multiplier assumes a constant value of 1 (logarithm is 0). Also, the increase and decrease of the multiplier value is more perceivable for values of γ close to 0.

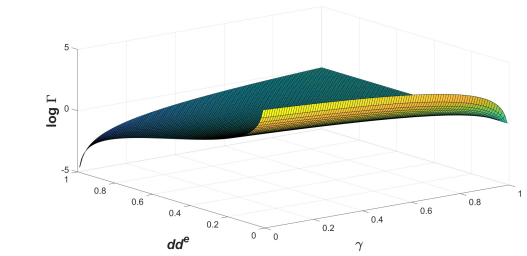


Figure 6: Evolution of $log(\Gamma)$ regarding the edge desirability and parameter γ

The anytime capability in the original FADPRM is realized through the parameter β, which establishes a balance between fast-solution search and best-solution search. The anytime parameter influences the selection of nodes from the *OPEN* list and, therefore which ones should be expanded. In other words, it implements an anytime capability through state expansion. To improve the anytime behavior of the algorithm, an anytime-based sampling strategy is proposed highly influenced in the approach taken by the anytime version of RRT [3]. Thus, the sampling strategy expression proposed (Equation 5) recurs to β as the tuning knob between exploration and anytime methodologies. The attribute `bias` represents the percentage of expanding the current `waypoint` towards the goal by a distance equal to `maxDistance`.

$$r = RandomNumber(0,1)$$

$$S_{strategy}(r) = \begin{cases} S_{goal} & \text{if } r \leq \texttt{bias} \\ S_{exploration} & \text{if } r > \texttt{bias} \wedge r > \beta \\ S_{anytime} & \text{if } r > \texttt{bias} \wedge r \leq \beta \end{cases} \tag{5}$$

Algorithm 3 Repairable Anytime Dynamic PRM

```
1:  procedure UpdateState(s)
2:      if search(s) ≠ counter AND search(s) ≠ 0 then
3:          if g(s) + h(s) < pathcost(search(s)) then
4:              h(s) ← pathcost(search(s)) − g(s)
5:              g(s) ← ∞
6:      else if search(s) = 0 then
7:          g(s) ← ∞
8:      search(s) ← counter

9:  procedure FindPath( )
10:     OPEN, CLOSED ← ∅
11:     insert s_start into OPEN with Key(s_start)
12:     while NoPathFound do
13:         remove s with max Key(s) from OPEN
14:         if Connect(s, s_goal) then
15:             return β-suboptimal path
16:         else
17:             ExpandNode(s)
18:             for all s' ∈ Neighbor(s) do
19:                 UpdateState(s')
20:                 g(s') ← g(s) + c(s, s')
21:                 insert s' into OPEN
22:             insert s into CLOSED
23:     counter ← counter + 1
24:     pathcost(search(s)) ← g(s_goal)

25: procedure RepairRoadmap( )
26:     Given G = (N, E), compute the function u(s), s ∈ N
27:     X ← toRepair · card(N)
28:     loop X times
29:         select node s, with the min u(s) from N
30:         N ← N \ {s}
31:         for all neighbors s' of s do
32:             E ← E \ {(s, s')}
33:             update neighbors s' of s

34: procedure Main( )
35:     RepairRoadmap( )
36:     counter = 1
37:     search(s_start), search(s_goal) ← 0
38:     β ← β_0
39:     UpdateState(s_start)
40:     UpdateState(s_goal)
41:     g(s_start) ← 0
42:     FindPath( )
43:     publish current β_0 − suboptimal solution
44:     while β ≠ β_F do
45:         if changes in edge costs are detected then
46:             ConsistencyProcedure( )
47:             decrease β or replan from scratch
48:         if β < β_F then
49:             increase β
50:         FindPath( )
51:         publish current β − suboptimal solution
```

$S_{exploration}$ intends to probe the whole environment and reach the goal in a very fast manner. On the other hand, $S_{anytime}$ operates differently, resorting to an `ELLIPSOIDAL` sampling strategy. If it's of interest only generating solutions cheaper than some upper bound value C_s, then that upper bound can be used to influence the sampling process. Therefore, rather than randomly sampling the entire `environment`, sampling is restricted just to those areas that could possibly provide a solution satisfying the upper bound.

As a multiple-query approach the roadmap computed by PRM should be amenable to an uncountable amount of different start and goal configurations. However, if every time a node is extracted from *OPEN*, a new sample is generated, the roadmap will be continuously growing. A huge number of nodes and consequently `edges` can be seen more as a burden, than an advantage for computational efficiency purposes. For this reason, a repairing procedure is proposed to restore the roadmap from an oversampling and saturation state. The RepairRoadmap method removes nodes that are not useful or that don't contribute to capturing the connectivity of the roadmap. Thus, a utility measure u is defined and the correspondent values are assigned to each and every node.

A good measure should have in high consideration isolated nodes with few neighbors. On the contrary, overpopulated zones are likely to host some useless nodes. The proposed expression for the utility measure (Equation 6) takes into account the cardinality of the set of neighbors $N(s)$.

$$u(s) = \frac{1}{1 + N(s)} \tag{6}$$

Algorithm 3 shows the pseudocode of the planner.

5. Results

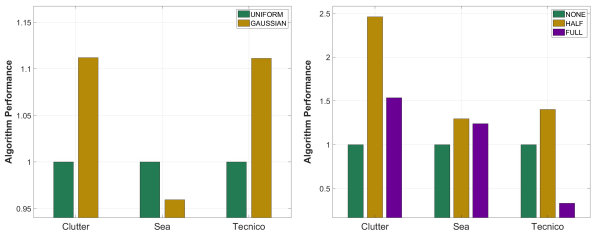
To draw solid and coherent conclusions from the results a proper testing setup should be established. For this reason, the basis for all testing procedures are three created testing environments characterized as follows: *Clutter* (Figure 7a), *Sea* (Figure 7b) and *Tecnico* (Figure 7c). The planners shall be evaluated according to three criteria: the cost of the solution, computation time, and a performance measure (Equation 7) which combines both. The mean (sketched as $\times$) and standard deviation (represented by a vertical bar) are two of the most important and revealing statistics indicators and therefore were chosen to assess the same two criteria. The performance measure p is inversely proportional to the mean of the normalized cost $\bar{c}$ and to the mean of the normalized time spent $\bar{t}$.

$$p(ALG) = \frac{1}{\bar{c} \cdot \bar{t}} \tag{7}$$

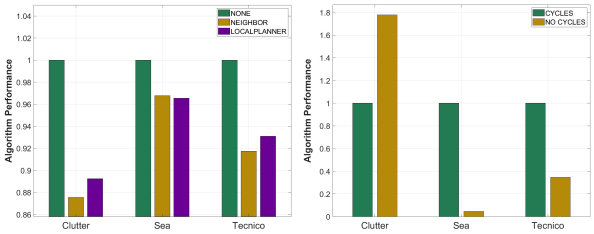
5.1. Rigid PRM

The results obtained for the two implemented `Sampling` strategies are shown in Figure 8a. The `GAUSSIAN` strategy yields better results than the `UNIFORM` regarding obstructed environments (*Clutter* and *Tecnico*), mainly due to a faster roadmap creation. However, the `GAUSSIAN` approach fails more frequently in finding a path − 82 queries in 7500 iterations, while `UNIFORM` only failed 3 times. As for the *Sea* environment, the `UNIFORM` strategy has a better performance. The lack of `waypoints` in the extremities of the environment could be a reason behind a higher cost for `GAUSSIAN`.

The performance of the algorithm for the different collision delays modes are represented in Figure 8b. In overall, the `HALF` mode is the superior collision delay technique. Even with more sampled `waypoints`, it has the lowest mean computation time for all three scenarios. It performs better than `NONE` because it doesn't execute upfront edge collision checks. Most of the collision checking computation time is spent on `edges` because there are more and its discretization process is lengthy. Also, since connections between collision-free nodes are more likely to be valid, it doesn't spend nearly as much time as `FULL` in correcting the roadmap.



(a) Sampling Strategy (b) Collision Delay

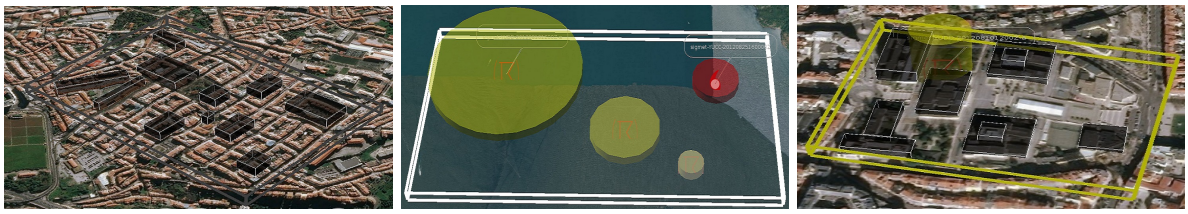


(c) Enhancement Mode (d) Presence of Cycles

Figure 8: Algorithm performance

Figure 8c illustrate the results collected for the several `EnhancementModes`. The results are pretty clear: the inclusion of an Enhancement Step is not beneficial, at least for the two implemented modes. In other words, it is better to randomly sample the total number of `waypoints` throughout the Construction Step. Perhaps, if the ratio of `waypoints` to be created during the Enhancement Step was tuned, better results could be achieved.

The results obtained for PRM with and without cycles are pictured in Figure 8d. The original version of PRM outputs generically paths with higher cost. The goal and start must be in the same component in order to exist a valid solution, and at the same time, that path is unique as a result of the same component rule. A particularly high value is verified for the *Sea* environment because in certain cases the only valid path traverses the SWIM

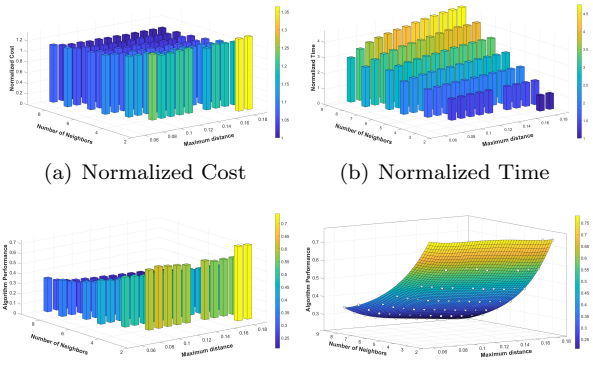


(a) *Clutter* environment (b) *Sea* environment (c) *Tecnico* environment

Figure 7: Testing environments

`obstacles`. On the other hand, this approach generates a much lower amount of `edges`, and therefore the roadmap creation and queries are processed at a faster pace. According to the defined performance measure, the original PRM has a superior behavior only in the *Clutter* scenario. However, as part of the SAFCS framework, which inherently includes manifold SWIM `obstacles`, the detected flaw makes this version of the algorithm almost unusable. Original PRM is only guaranteed to perform better in environments with no SWIM `obstacles` and in that case smoothing techniques should be applied to improve the quality of the path.

The bounded-degree neighborhood selection method of the standard `RigidPRM` is subject to study (Figure 9) in the *Tecnico* scenario. The values for `maxNeighbors` ranged from 2 to 9, while `maxDistance` was successively incremented by 1% from 5% to 19% of the longest internal diagonal of the environment.



(a) Normalized Cost (b) Normalized Time

(c) Algorithm Performance (d) Surface fit

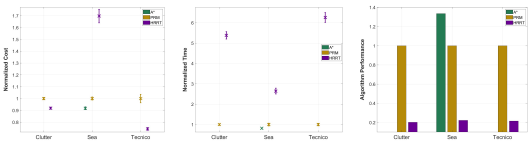
Figure 9: Results relative to different values of `maxDistance` and `maxNeighbors`

The values of the normalized costs have an expected evolution. An increase in the number of neighbors and in the maximum distance leads to a better solution (path with a lower cost). On the other hand, the computation time also increases and does so at a faster rate than the decrease of costs. It is worth to mention that the number of neighbors is the major factor influencing the computation time.

Considering the selected number of sampling iterations, the optimal connection parameters, according to the data collected and to the chosen performance measure, are `maxNeighbors`= 3 and `maxDistance`= 18% of the longest internal diagonal of the environment.

The fitting surface (Figure 9d) was obtained as a result of the Matlab® function *fit* for a fourth-degree polynomial (*poly44*) in x (Maximum distance) and y (Number of Neighbors). Since the performance measure (Equation 7) does not take into account the number of times the algorithm succeeds or fails, the fitting surface suggests low values for the connection parameters as the best approach. However, the parameters cannot be lowered indefinitely, as it will result in the failure of an excessive number of queries.

In Figure 10, the data comparing the performance of A*, PRM and heuristic RRT (HRRT) is shown. The A* comparison was limited to the *Sea* scenario, since its current implementation in SAFCS does not support the untraversable obstacles present in the other scenarios.

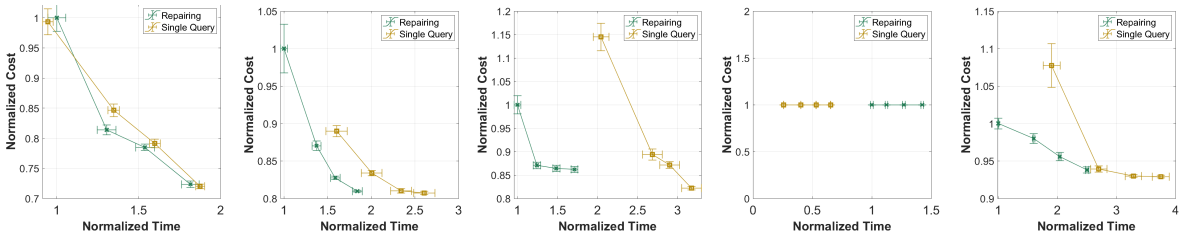


(a) Normalized Cost (b) Normalized Time (c) Algorithm Performance

Figure 10: Comparison of results between A*, PRM and HRRT

Regarding the results obtained for the *Sea* scenario, the optimal algorithm is A*. HRRT performance is highly affected by an extraordinarily high cost in the *Sea* environment, due to its inability to, in some cases, avoid SWIM `obstacles`.

PRM behaves better than RRT in obstructed environments, considering a multiple query approach. In the PRM planner, if a `waypoint` A is created in a difficult zone and is not connectable to any other,



(a) Query 1 (b) Query 2 (c) Query 3 (d) Query 4 (e) Query 5

Figure 11: Normalized costs and times of 5 queries for a repairing and a single query PRM

it is still stored in the set of `waypoints`. Later on, other nodes can be sampled in the same zone and establish connections to `waypoint` A. On the contrary, RRT creates `waypoints` by extending the nearest one. Even if the new `waypoint` is valid, if the connection between them is not, then all the process is discarded.

A further comparison point regards the mean time per query excluding the roadmap creation in PRM. Even though A* and PRM use the same planner to search for a path, the mean query time is lower for PRM. This discrepancy in values can only be explained with the different implementations for the same methods. On the other hand, if only one query is processed by each iteration of roadmap, then a single-query approach is established. RRT is a planner especially designed to cope with single-query problems and outperforms PRM in that case

5.2. Flexible approach

The results obtained for the *Sea* environment are pretty clear: the inclusion of the anytime sampling strategy yields better results. In overall, it optimizes the path in a quicker manner than the non-anytime sampling version. The improvement is more pronounced in the $2nd$ and $3rd$ iterations of the solutions, where the anytime strategy shows its influence. Moreover, the replanning version (FAD-PRM) has a much better temporal performance. Nonetheless, its best solution has a slightly lower quality than the ones found in other algorithms.

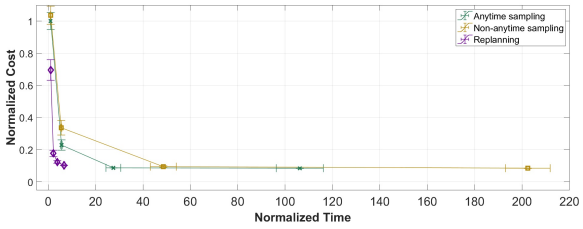


Figure 12: Comparison of results between several variations of flexible PRM

To validate the proposed Repair Step, the performance of the algorithm is tested in three conditions: building the roadmap from scratch in every query (Single-query approach), repairing 30% (`toRepair`=0.3) of the roadmap in the beginning of each query and maintaining the previous roadmap (`toRepair`=0). If the same roadmap is maintained from query to query, then the processing time becomes extraordinarily high in virtue of an enormous amount of `waypoints` and `edges`.

Figure 11 depicts the results obtained for the repairing version and the single query approach. The performance is quite similar for the first query. However, in the subsequent queries the repairing approach produces better results specially regarding the computation time. The fourth query represents a configuration of start and goal `positions` that are very close to each other and therefore the path is fully optimized on the first iteration of the algorithm. In this case, the single-query method has a better performance, since it deals with fewer `waypoints` and `edges` and doesn't spend time repairing the roadmap.

6. Experimental Work

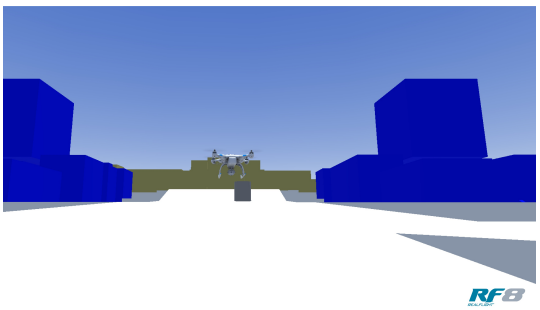


Figure 13: RealFlight 8® Instituto Superior Tcnico environment

The SITL (software in the loop) simulator is a build of the *ArduPilot* which enables the creation and testing of several vehicles. To generate a simulated quadcopter, DroneKit-SITL must not only be initiated with a valid *ArduCopter* version but also with proper aircraft parameters. RealFlight 8® is regarded as one of the best RC flight simulators and allows the customization of airports (Fig-

ure 13). The simulated aircraft, the flight simulator and the SAFCS framework interface with each other through a `DronekitDatalink` connection and a `Droneconnect Server`.

Only qualitative results can be withdrawn from the simulations performed. The SAFCS was able to successfully receive real-time information about the aircraft. Regarding the tests executed, the aircraft was able to follow the computed path, while avoiding untraversable obstacles.

Concerning the experimental work with a real aircraft, the flight tests were never allowed due to the lack of a *Special Flight Operations Certificate* (SFOC) from Transport Canada.

7. Conclusions

The primary goal behind the development of this thesis was achieved: the integration of the PRM family into the SAFCS framework. Several new classes and features were added to SAFCS.

The implementation of the algorithms praised for a great level of configurability including many parameters and techniques. `RigidPRM` encases several versions of PRM and is easily extendable. Regarding the flexible approach, novel features and improvements were suggested. Finally, an anytime and dynamic planner was successfully implemented. Even though the implementation of all sorts of techniques was quite straight-forward, deducing the most advantageous approach wasn't trivial. Indeed, the parameters are highly influenced by each other and it would require an overwhelming amount of data to figure out the optimal parameters.

The rigid approach, which builds the roadmap *a priori*, thrives in environments highly populated by static and untraversable obstacles. Since it stores a roadmap, PRM is more appropriate whenever multiple queries (in the same environment) shall be answered. Even though it doesn't outperform A*, the creation of a roadmap has the upper hand relatively to the RRT expansion strategy, specially in obstructed environments. From the results obtained, one can conclude that PRMs are suitable for aircraft 4D path planning.

The modifications suggested for the flexible approach of PRM, namely the anytime sampling strategy and the Repair Step, proved to improve the performance of the algorithm.

References

[1] Khaled Belghith, Froduald Kabanza, and Leo Hartman. Randomized path planning with preferences in highly complex dynamic environments. *Robotica*, 31(8):1195–1208, 2013.

[2] R. Bohlin and L. E. Kavraki. Path planning using lazy prm. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings*, volume 1, pages 521–528 vol.1, April 2000.

[3] D. Ferguson and A. Stentz. Anytime rrts. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5369–5375, Oct 2006.

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

[5] Stephan Heinemann. *Smart Autoflight Control Systems - Dissertation Proposal*. PhD thesis, University of Victoria, 2016.

[6] Stephan Heinemann, Hausi A. Müller, and Afzal Suleman. Smart autoflight control systems. In *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*, CASCON '14, pages 343–346, 2014.

[7] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, 30(7): 846–894, June 2011.

[8] L. E. Kavraki, M. N. Kolountzakis, and J. . Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, 14(1):166–171, Feb 1998.

[9] Lydia Kavraki, Petr Svestka, Jean claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE International Conference on Robotics AND Automation*, pages 566–580, 1996.

[10] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, New York, NY, USA, 2006.

[11] Tomas Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32:108–120, 1983.

[12] Xiaoxun Sun, Sven Koenig, and William Yeoh. Generalized adaptive a*. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS, pages 469–476, 2008.

[13] Liang Yang, Juntong Qi, Dalei Song, Jizhong Xiao, Jianda Han, and Yong Xia. Survey of robot 3d path planning algorithms. *J. Control Sci. Eng.*, 2016:5–, March 2016.