GPU-Accelerated RRT for Flight Planning Considering Ensemble Forecasting of Thunderstorms

Eduardo Andrés, Daniel González-Arribas,
Manuel Sanjurjo-Rivo, Manuel Soler
Department of Bioengineering and Aerospace Engineering
Universidad Carlos III de Madrid
Leganés, Spain
eandres, dangonza, msanjurj, masolera@ing.uc3m.es

Maryam Kamgarpour
Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
maryamk@ece.ubc.ca

***Abstract*—The intrinsic uncertainty of thunderstorms poses a major threat for flights, as it jeopardizes the safety of the passengers and the airframe. In this paper, we enhance the performance of the scenario-based rapidly-exploring random tree star (SB-RRT*), a methodology for aircraft trajectory optimization that considers ensemble-based products for weather phenomena. Through parallelization on graphical processing units the simulation time is reduced substantially as required by practical settings. We test the method considering a unicycle model of an aircraft flying between two state configurations at constant flight level and airspeed. Lastly, we compare the parallelized and the original SB-RRT* under the same conditions. Results show how the new version is able to reduce computational times in 3 orders of magnitude with respect to the original and provide safe and close-to-optimum solutions in near-real time.**

***Keywords*—Optimal path planning, Sampling-based algorithms, Uncertain thunderstorm avoidance, Parallel programming.**

I. Introduction

Uncertainties related to convective weather represent an important concern for the Air Traffic Management (ATM) system, as they cause safety, capacity and efficiency disruptions. In 2019, 21% of the delays in Europe were attributed to weather [1]. Specifically, thunderstorms constitute a considerable risk, with hail, turbulence or wind shear as possible associated phenomena. Therefore, to avoid them is a key requirement to guarantee passenger's comfort and aircraft's structural integrity. Thunderstorms evolve in rapid timescales and their location and timing are hard to predict with certainty. For these reasons, new algorithms for flight scheduling must take stochasticity into account, incorporate new weather measurements and be able to produce solutions in short time periods. The objective of this paper is the design of a path planning methodology able to provide optimal and safe trajectories for aircraft considering uncertain thunderstorms and to operate in near-real time.

The aforementioned problem can be formulated as a trajectory optimization subject to a stochastic environment. We recommend referring to [2] for a good survey on the topic. However, despite the wide variety of methodologies, only a few works have dealt with the problem of aircraft flying in regions of uncertain thunderstorm development.

A first class of methods are those based on optimal control techniques. On one hand we find the so-called stochastic Reach-Avoid [3], based on Dynamic Programming (DP), able to deal with uncertainties in aircraft motion and thunderstorm encounters. The main disadvantage from DP methodologies is that they are subject to the "Curse of Dimensionality"; computational time can be prohibitive as it scales with the dimension of the state space and the discretization. On the other hand, the optimal control problem can transcribed into a nonlinear programming problem. In [4], the authors solve the motion of a 3 degrees-of-freedom model of aircraft with the uncertain evolution of convective weather. Its main drawback, the sensitivity to the initial guess, is overcome with a randomized initialization, and in consequence the algorithm is able to provide a portfolio of solutions. Although both approaches can be used in complex settings with nonlinear dynamics, its ability to handle operational constraints is limited.

An alternative way to address the problem are metaheuristic methodologies, which, contrary to the techniques mentioned above, are based on a exhaustive exploration and exploitation of the state space to get near-optimal solutions. Metaheuristic algorithms were previously used for deterministic weather avoidance [5] and flight planning under different sources of uncertainty [6], [7]. In [8], we presented the scenario-based rapidly-exploring random tree star (SB-RRT*), a RRT*-based algorithm able to provide safe, continuous and close-to-optimum trajectories in uncertain environments described by ensemble-based products. The main drawback of this methodology is the large computational time (order of days), which makes it unusable for practical purposes.

The aim of the present paper is to update the SB-RRT* with a parallelization of the most time consuming step: the safety check. The algorithm makes an exhaustive use of that function, which increases with the number of iterations, thus a reduced computational time associated to each call is necessary for

fast simulations. We use parallel computing techniques on graphical processing units (GPUs) to improve significantly the running times per safety check. In consequence, this methodology is able to reduce the original prohibitive computational time to minutes (or even seconds). As an illustrative example, we test the algorithm assuming a unicycle model of an aircraft flying straight between states and maintaining constant altitude and airspeed.

The paper is structured as follows. In first place, we frame our approach in Sec.II. Then, we present the SB-RRT* algorithm and the parallelization in Sec. III. We test the update and compare it to the initial implementation in Sec. IV. Lastly, we outline the main conclusions and possible future works in Sec. V.

II. Problem Framework

Aviation stakeholders consider thunderstorms to be one of the most dangerous events during flights and it is preferred to avoid them whenever is possible. Nevertheless, it is difficult for pilots to find the safest route in stormy regions, entailing delayed and diverted flights, as well as an increase in fuel consumption and total costs. The reasons for this are threefold:

- First, thunderstorms are uncertain phenomena that happen on relatively fast timescales (around 30 minutes) and are hard to predict.
- Second, for flight planning purposes, pilots make use of weather charts, which are not updated during a flight and become obsolete. Relevant weather changes are communicated verbally by the air traffic controller or other aircraft. Weather charts are obtained through numerical weather prediction (NWP) models some days in advance and do not have sufficient spatiotemporal resolution to capture convective phenomena (such as thunderstorm birth and growth). Additionally, low resolution products that cover large regions such as Convective Significant Meteorological Information (SIGMET) are used by pilots to decide which regions to avoid.
- Lastly, during a flight, the main source of recent weather information is the onboard radar. Weather radars present multiple constraints, the range is often limited to 150 nm (around 20 minutes of flight) and the scan region is restricted to the aircraft front. There is no information about lateral regions, which might be relevant for possible diversions.

There exist many ongoing efforts to improve the weather data available for aviation. In first place, the use of ensemble prediction systems (EPS). An EPS is a NWP forecasting technique used to characterize atmospheric uncertainty. It provides, typically, between 10 and 50 numerical weather forecasts considering small perturbations in the models and the parameters [9]. Existing EPS are not yet able to capture the phenomenology behind convective events. Nonetheless, in the near future (5 to 10 years), NWP methods are expected to evolve towards very short term ($\sim$1 h) and very high resolution ($\sim$100 m) convective-permitting EPS able to better capture thunderstorms. Consequently, upcoming planning algorithms must be able to work with ensemble-based weather predictions that would eventually be able to forecast thunderstorms.

A second effort is focused on the ground-air link of data and the fusion with onboard information. The main idea is to combine NWP forecasts, with radar, satellite and other additional observations and display the results on the cockpit [10]. Despite the fact that uplink systems have been successfully tested in the past in projects such as FLYSAFE [11] or eFlightOps [12], the real implementation in commercial aircraft is still under research. Aviation is subject to strict regulation and certification processes that need to be overcome before these systems are ready to be included in primary flight displays. However, the representation of such data on complementary devices (electronic flight bags) would provide pilots additional information and more time to react to thunderstorm evolution, minimizing deviations from the planned trajectory and hence saving fuel. An example of that type of technology is eWAS Pilot [1], an app that provides pilots real time weather information from several sources through Wifi or 4G connections.

The aim of the SB-RRT* is to be used together with the aforementioned systems, suggesting possible diversions from the flight plan to overcome updated weather events. An sketch is represented in Fig.1 and an example of possible ensemble forecast over Europe is shown in Fig.2.

III. Parallelized Scenario-Based RRT*

In this section, we present the SB-RRT* algorithm, introduced in [8], and enhance its computational performance by using parallelization techniques on GPUs.

A. Environment Definition

We define the flight space as $\mathcal{X} \subset \mathbb{R}^{d_x}$, with d_x equal to 2 or 3, depending on whether the motion occurs in a 2D plane or a 3D volume. We also define the unsafe set that includes thunderstorms as $X_{unsafe} \subset \mathcal{X}$ and the safe-to-fly regions as $X_{safe} = \mathcal{X} \setminus X_{unsafe}$.

In this scenario-based approach we consider X_{unsafe} to be a weather ensemble forecast formed by N_{sc} possible members or realizations. Each member is formed by a group of storm cells, each of them denoted by C_l^j. We refer to a particular member of the ensemble with j and a storm cell from the j-th member with l, where $j = 1, \dots, N_{sc}$ and $l = 1, \dots, N_c^j$. The number of storm cells N_c^j is not constant between scenarios. We assume that C_l^j is a deterministic closed region, fully included in $\mathcal{X}$ and delimited by a polygon or an ellipse. For this reason, we can check if a path intersects any C_l^j by means of geometric operations.

We define the j-th member of the ensemble X_{unsafe}^j as the union of all the storm cells from that member. Then,

$$X_{unsafe}^j = \bigcup_{l=1}^{N_c^j} C_l^j. \tag{1}$$

[1] http://www.ewas.aero/product/ewas-pilot

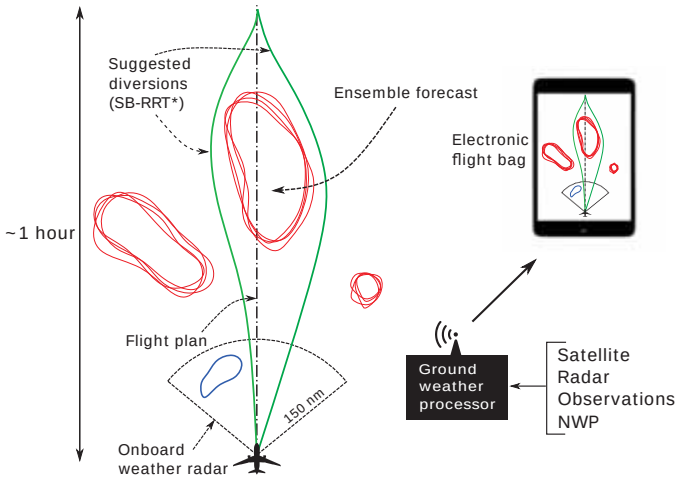


Figure 1. Representation of trajectories suggested by the SB-RRT* (green lines) on a electronic flight bag. Storms detected by the onboard weather radar are represented in blue. An ensemble forecast of 4 members provided by ground-air data link is represented in red.

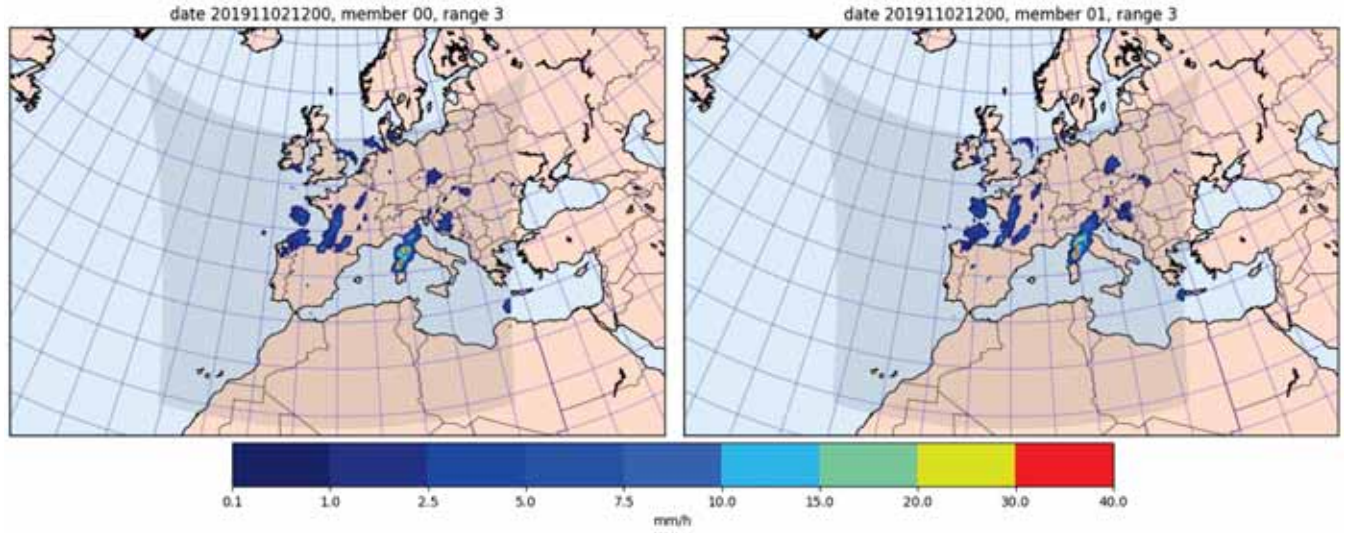


Figure 2. Example of ensemble weather forecast (showing just 2 members) obtained with the Short-Term Ensemble Prediction System (STEPS) methodology applied to observations that merge EUTMENET OPERA radar composite and SAF RDT satellite data.

The ensemble X_{unsafe} is the set that includes the different members X^j_{unsafe}, hence

$$X_{unsafe} = \{X^1_{unsafe}, \ldots, X^{N_{sc}}_{unsafe}\}. \quad (2)$$

We assume, without loss of generality, that each member of the ensemble happens with the same probability. That is, $\Pr(X_{unsafe} = X^j_{unsafe}) = 1/N_{sc}, \forall j$. However, the formulation can be extended to consider ensemble members with different weights.

B. SB-RRT* Algorithm

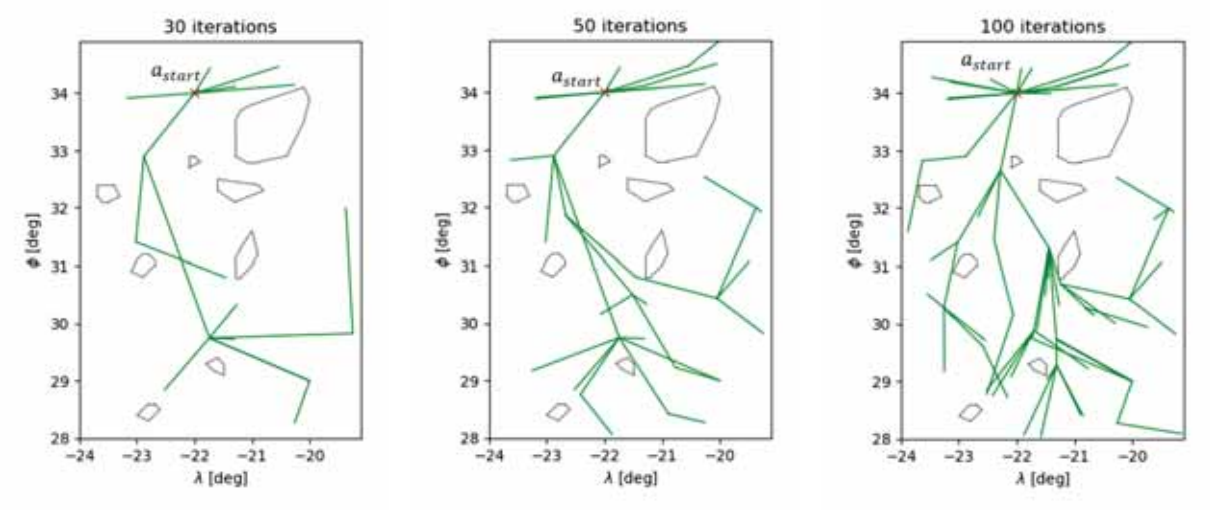
We define the aircraft state space as $\mathcal{S} \subset \mathbb{R}^{d_s}$. In general, $d_s \geq d_x$, as $\mathcal{X}$ only represents possible aircraft positions, while $\mathcal{S}$ might include dynamical variables such as fuel consumption, velocity or attitude angles. We also define the control space as $\mathcal{U} \subset \mathbb{R}^{d_u}$. The aircraft dynamics is represented by a state vector $\mathbf{s} \in \mathcal{S}$ that changes according to a transition equation,

$$\dot{\mathbf{s}} = f(\mathbf{s}, \mathbf{u}), \quad (3)$$

where $\mathbf{u} \in \mathcal{U}$ is the control input.

The SB-RRT* belongs to the category of incremental sampling-based algorithms and its objective is to build a graph $\mathcal{G} \subset \mathcal{X}$, such that:

- It is an explicit representation of X_{safe}.
- It is created iteratively, it expands through a process of random sampling of the state space $\mathcal{S}$, setting feasible connections between pairs of safe samples.
- It is asymptotically optimal, converging almost surely to a trajectory of minimum cost if the maximum number of iterations, $MaxIter$, is sufficiently large.



(a) 30 iterations. (b) 50 iterations. (c) 100 iterations

Figure 3. Example of SB-RRT* evolution with 30, 50 and 100 iterations.

- It is a single-query algorithm that grows according to an origin and destination defined beforehand.

The graph $\mathcal{G}$ is a combination of two elements: nodes and edges. We define the node $a \in \mathcal{X}$ as the projection of a randomly sampled state $s \in \mathcal{S}$ in $\mathcal{X}$. That means that a is an object represented by a position in $\mathcal{X}$, and might also include additional variables, such as velocity or attitude values. We also define the edge $e \subset \mathcal{X}$ as the trajectory that results from the integration of Eq. (3) between a pair of nodes. The sets $\mathcal{A}$ and $\mathcal{E}$ represent, respectively, the groups of nodes and edges successfully included in $\mathcal{G}$.

We define the initial and target states, $\mathbf{s}_{start}$, $\mathbf{s}_{goal} \in \mathcal{S}$, and the associated nodes a_{start} and a_{goal}. The graph is initialized with a_{start} and grows trying to reach a_{goal}. During each iteration, a node and an edge are tested for safety in order to be added to $\mathcal{A}$ and $\mathcal{E}$. Failing this check means rejection of the sample and that possible connection. The SB-RRT* algorithm is detailed in Algorithms 1-3 and is illustrated in Fig. 3.

Algorithm 1 $\mathcal{G} = (\mathcal{A}, \mathcal{E}) \leftarrow$ SB-RRT*(a_{start})

$\mathcal{A} \leftarrow a_{start}$, $\mathcal{E} \leftarrow \{\emptyset\}$;
while $k < MaxIter$ **do**
  $a_k \leftarrow$ *RandomSample*();
  $a_{nearest} \leftarrow$ *NearestNode*$(a_k, \mathcal{A})$;
  $e_{nearest} \leftarrow$ *Steer*$(a_{nearest}, a_k)$;
  if *Safe*$(e_{nearest})$ **then**
    $A_{near} \leftarrow$ *Near*$(a_k, \mathcal{A})$;
    $a_{parent}, e_{parent} \leftarrow$ *BestParent*$(a_k, a_{nearest}, e_{nearest}, A_{near})$;
    $\mathcal{A} \leftarrow$ *Add*(a_k), $\mathcal{E} \leftarrow$ *Add*(e_{parent});
    $\mathcal{G} \leftarrow$ *Rewire*(a_k, A_{near});
  end if
end while
return $\mathcal{G}$

Algorithm 2 $a_{parent}, e_{parent} \leftarrow$ *BestParent*$(a_k, a_{nearest}, e_{nearest}, A_{near})$

$a_{parent} \leftarrow a_{nearest}$, $e_{parent} \leftarrow e_{nearest}$;
$c_{min} \leftarrow$*Cost*$(a_{start}, a_{nearest})$ + *Cost*$(a_{nearest}, a_k)$;
for $a_{near} \in A_{near}$ **do**
  $e_{near} \leftarrow$ *Steer*(a_{near}, a_k);
  if *Safe*(e_{near}) **then**
    $c_{near} \leftarrow$ *Cost*(a_{start}, a_{near}) + *Cost*(a_{near}, a_k);
    if $c_{near} < c_{min}$ **then**
      $a_{parent} \leftarrow a_{near}$, $e_{parent} \leftarrow e_{near}$;
      $c_{min} \leftarrow c_{near}$;
    end if
  end if
end for
return a_{parent}, e_{parent}

Algorithm 3 $\mathcal{G} \leftarrow$*Rewire*(a_k, A_{near})

for $a_{near} \in A_{near}$ **do**
  $e_{near} \leftarrow$ *Steer*(a_k, a_{near});
  $c_{near} \leftarrow$ *Cost*(a_{start}, a_k) + *Cost*(a_k, a_{near});
  if *Safe*(e_{near}) **and** $c_{near} <$*Cost*(a_{start}, a_{near}) **then**
    $a_{parent}, e_{parent} \leftarrow$ *Parent* (a_{near})
    $\mathcal{E} \leftarrow$ *Remove*(e_{parent})
    $\mathcal{E} \leftarrow$ *Add*(e_{near})
  end if
end for
return $\mathcal{G}$

The following functions are required to grow the tree and are common to many sampling-based algorithms:

- *RandomSample*: it creates a node a_k through random sampling of $\mathcal{S}$ (typically with a uniform distribution).
- *NearestNode*: given a node a_k, it returns the closest

node $a_{nearest} \in \mathcal{A}$, according to a metric, e.g., Euclidean distance or Dubins path length.
- *Steer*: it propagates Eq. (3) between two nodes a and a' minimizing a cost function, e.g. distance, time or fuel consumption. This can be achieved either by solving an optimal control problem to get the required control $\mathbf{u}$ or by using precomputed optimal solutions such as Dubins curves. The resulting path is an edge e.
- *Safe*: this function checks if an edge e is safe before including it in $\mathcal{E}$ and it is the objective to be parallelized in this work.
- *Cost*: it calculates the cost of the path between two nodes a and a'. It defines the objective to be minimized.
- *Parent*: for any node a it returns its parent node (the previous node to which is connected) $a_{parent} \in \mathcal{A}$ and the edge $e_{parent} \in \mathcal{E}$ connecting both. The node a is called child of a_{parent}
- *Near*: it returns the set of nodes $A_{near} \subseteq \mathcal{A}$ within a ball centered at a_k with radius $\gamma(\frac{\log \text{card}(\mathcal{A})}{\text{card}(\mathcal{A})})^{\frac{1}{d_x+1}}$ (see [13]), where γ is a constant required for optimality and $\text{card}(\mathcal{A})$ represents the cardinality of $\mathcal{A}$ during the corresponding iteration.
- *Add*, *Remove*: these functions include or remove nodes and edges from $\mathcal{G}$.

Algorithm 1 includes a typical RRT* methodology to grow a graph $\mathcal{G}$ by connecting randomly chosen states to the closest nodes in $\mathcal{G}$. Then, by using *BestParent* and *Rewire* functions, detailed in Algorithms 2 and 3, the internal connections in $\mathcal{G}$ are restructured. The result is a graph that optimizes the cost of the paths towards each node in $\mathcal{A}$.

The main feature from the SB-RRT*, and what differences it from other RRT*s, is the so-called *Dynamic Risk Allocation*, a methodology to test if new connections are safe considering scenario-based uncertainties. According to the formulation in [8], we define any path that connects the initial node a_{start} to any state obtained randomly as P. Each P is a sequence of N edges, $P = \{e_1, \ldots, e_N\}$. The Dynamic Risk Allocation states that any path P in the tree, including the solution, must be safe with a safety margin $\epsilon \in [0, 1]$. That is,

$$\Pr(P \in X_{safe}) = \Pr\left(\bigwedge_{i=1}^{N} e_i \in X_{safe}\right) \geq 1 - \epsilon. \quad (4)$$

The big wedge operator ($\wedge$) represents a logical conjunction. By means of Boole's inequality, the previous condition can be conservatively satisfied as follows,

$$\Pr(P \cap X_{unsafe}) \leq \sum_{i=1}^{N} \Pr(e_i \cap X_{unsafe}) \leq \epsilon. \quad (5)$$

The probability $\Pr(e_i \cap X_{unsafe})$ is estimated with the proportion of ensemble members intersected by e_i. As the tree is growing, any new connection must verify Eq. (5), so that any path can be considered safe. An example of the method considering $\epsilon = 0.1$ is illustrated in Fig. 4 with two different

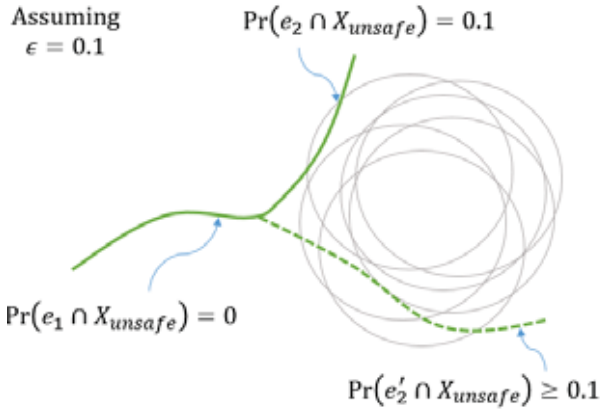


Figure 4. Illustration of DRA considering the two possible alternatives: acceptance of an edge that satisfies Eq. (5) and rejection of an edge that violates it (dashed line).

edges, e_2 and e_2'. The application of Eq. (5) to both possible paths results in:

$$\Pr(e_1 \cap X_{unsafe}) + \Pr(e_2 \cap X_{unsafe}) = 0.1 \leq \epsilon,$$

and,

$$\Pr(e_1 \cap X_{unsafe}) + \Pr(e_2' \cap X_{unsafe}) \geq \epsilon.$$

On one hand, e_2 is accepted, as the total sum of probabilities is not over ϵ and the tree could keep growing in that direction. On the other hand, e_2' is rejected, as the aggregated risk violates the value of 0.1. That way risk is assigned non-uniformly to the different edges as the SB-RRT* is growing.

C. Parallelization

The safety checks resulting from Eq. (5) and the calculation of each $\Pr(e_i \cap X_{unsafe})$ were the main bottleneck from the initial SB-RRT* implementation. The process was done sequentially, calculating the intersection of an edge with each ensemble member X^j_{unsafe}. Additionally, if each member has several storm cells C^j_l, the intersections were also determined sequentially. The total number of intersections to be obtained each time *Safe* is called is $\sum_{j=1}^{N_{sc}} N^j_c$, hence the computational time required for checking one edge grows linearly with the number of ensemble members and the number of storm cells per member. As the algorithm requires an extensive use of this function (lines 6, 5 and 4 in Algorithms 1-3, respectively) that increases with the number of nodes, the total execution time grows exponentially with the maximum number of iterations.

The objective of this work is to address this issue by means of parallelization techniques, using CUDA [14], a platform for general computing on GPUs. To calculate the value of $\Pr(e_i \cap X_{unsafe})$, all the storm cells C^j_l from any ensemble member are handled in parallel. That way, the previous $\sum_{j=1}^{N_{sc}} N^j_c$ steps are reduced to 1, and *Safe* function becomes independent of both the number of members in the ensemble and the number of storm cells. The performance of this new approach is analyzed in the case study.

IV. Case Study

In this section, we compare the parallelized version of the SB-RRT* to our previous implementation [8]. We consider a unicycle model of an aircraft, which flies straight between nodes, and keeps constant altitude and airspeed.

A. Problem setting

In order to test the parallelized SB-RRT*, we consider the flight region $X = [-24°, -19°] \times [28°, 35°]$ with longitude λ and latitude ϕ as state variables and constant flight level FL300. We simulate an ensemble-based environment with storm cells described by polygons. An example considering 20 ensemble members is shown in Fig. 5. For this example, it is assumed that no-fly regions do not change with time. However, during the tree propagation, the algorithm is able to consider ensemble members at different times to deal with moving cells. The objective consists in finding a safe trajectory that connects $\mathbf{x}_{start} = (-22°, 34°)$ and $\mathbf{x}_{goal} = (-20°, 29°)$ while minimizing total flight distance. A safety margin $\epsilon = 0.1$ is considered, hence the constraint for each path P is $\sum_{i=1}^{N} \Pr(e_i \cap X_{unsafe}) \leq \epsilon$ (see Eq. (5)). The computations are performed in a workstation equipped with an Intel Core i7-8550U CPU running at 1.80 GHz and a NVIDIA GeForce GTX 1050 GPU.

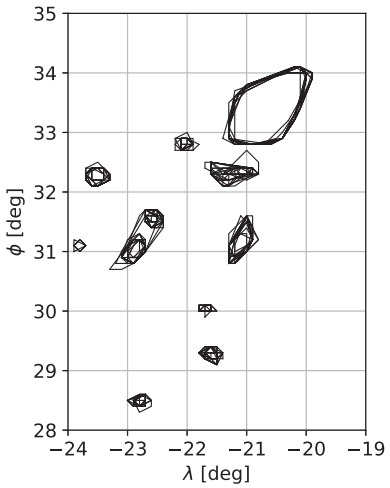


Figure 5. Example of ensemble-based environment with 20 members.

B. Computational performance

In this section, we analyze the effect of the parallelization on the *Safe* function on its own and as a part of the algorithm. Firstly, we include in Table I the computational time [2] required for checking the safety of one edge as a function of the number of scenarios, with and without GPU. As expected, it can be concluded that with no parallelization the required time for the safety check grows linearly with N_{sc}, whereas the parallel implementation involves almost constant time per check. We attribute this scaling to the fact that the GPU has enough parallel processing cores to handle all the storms in all the scenarios in a single kernel launch. Moreover, the time per call is reduced in 2-3 orders of magnitude depending on the value of N_{sc}.

[2] Average values considering 1000 different edges.

TABLE I. CPU vs. GPU. Computational time required for the intersection checking (per edge) as a function of the number of scenarios.

N_{sc}	With CPU (ms)	With GPU (ms)	$\text{time}_{\text{CPU}}/\text{time}_{\text{GPU}}$
5	720	1.22	590
10	1500	1.25	1200
20	2850	1.25	2280
50	6840	1.31	5220

In second place, we analyze how the new approach affects the computational time of a SB-RRT* simulation for $N_{sc} = 20$. In Table II, we show the time required by the simulation, in average, as a function of the maximum number of iterations $MaxIter$. Note that there are no results beyond 1000 iterations for the initial implementation as it would require several weeks.

TABLE II. CPU vs. GPU. SB-RRT* computational time as a function of the maximum number of iterations for $N_{sc} = 20$.

$MaxIter$	With CPU (h)	With GPU (s)
100	3.5	1.9
200	10.0	7.1
500	41.9	34.3
1000	120.6	122.9
2000	-	446.4
5000	-	2537.8

As it can be observed, the simulations that involved days to complete now are finished in a matter of minutes (or seconds). In particular, our previous upper bound, $MaxIter = 1000$, has reduced the computational time from 5 days to 2 minutes. Additionally, we include in Fig. 6 a continuous representation of computational time with the number of iterations for multiple simulations; the dependence is almost quadratic and if we double the iterations we might expect around 4 times more time.

C. Convergence and sensitivity of the solution

In this section we study the influence of the number of iterations on the algorithm convergence and the solutions considering $N_{sc} = 20$. The SB-RRT* is a heuristic algorithm that explores a region based on a random sampling process. Each simulation grows a different graph and approaches the solution in a new manner. An example of graph and solution after 5000 iterations is shown in Fig. 7. As it was stated beforehand, this methodology presents asymptotic optimality and would require an infinite number of iterations to obtain the trajectory of minimum cost. We analyze the number of iterations required for a close-to-optimum solution by running the same simulation 50 times. In Fig. 8 we represent the

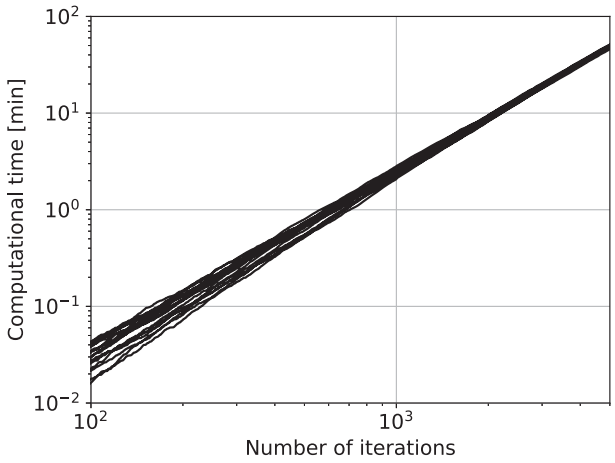


Figure 6. Computational time of the parallelized SB-RRT* as a function of the maximum number of iterations for $N_{sc} = 20$.

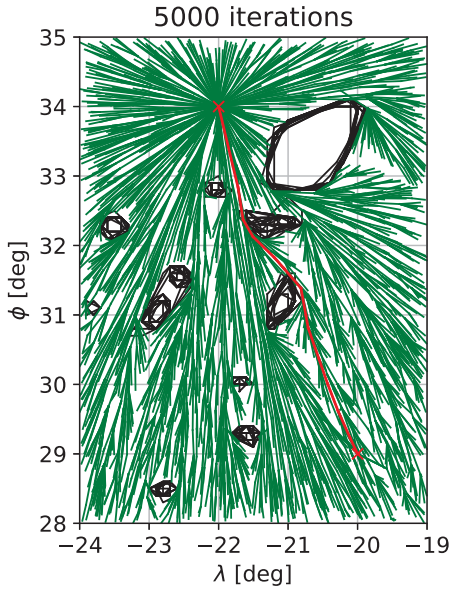


Figure 7. Example of SB-RRT* and solution after 5000 iterations with $N_{sc} = 20$.

relative error in flight distance with respect to its minimum value. [3] The results show that in half of the simulations this difference is 2.4% (or less) after 500 iterations and 1.1% after 1000. Moreover, we represent with different color bands the 66 and 90 percentiles. After 1000 iterations, a third of the solutions resulted in a difference above 1.2% and a 10% differed more than 2%. The greater difference was a 3.2%.

To conclude, we illustrate the sensitivity of the solutions to the maximum number of iterations. In Fig. 9 we show the solutions provided by 10 different simulations after 1000, 2000 and 5000 iterations. It can be observed in Fig. 9(c) that after 5000 iterations, or even more, the algorithm converges to the same solution in almost every case. However, due to the quadratic relation with the number of iterations, it requires around 25 and 6 times more computational time, respectively, compared to the simulations with 1000 and 2000 iterations. As it is shown in Figs. 9(a)-9(b), there exist a higher variability in the results in those cases, indicating the existence of different local optima. Considering the results shown in Fig. 8, after 1000 iterations all of the solutions are within a 3.2% margin in cost, and after 2000 the margin is expected to be tighter. This is indeed a positive fact, as we can obtain fast solutions and propose different near-optimal alternatives to pilots and air traffic controllers.

[3]The minimum flight distance was calculated with a simulation of 10^5 iterations.

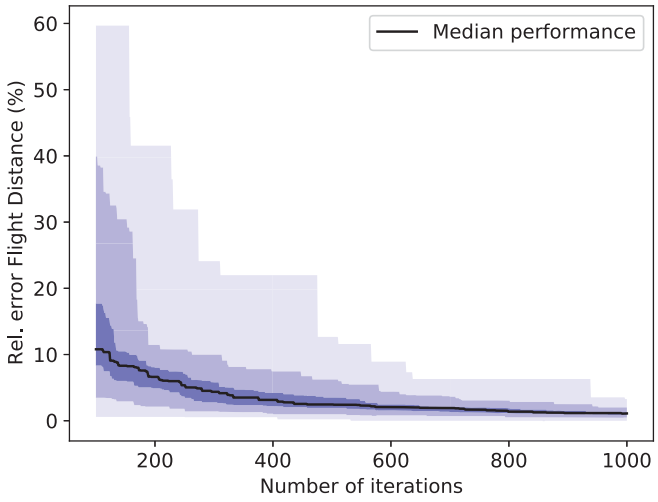
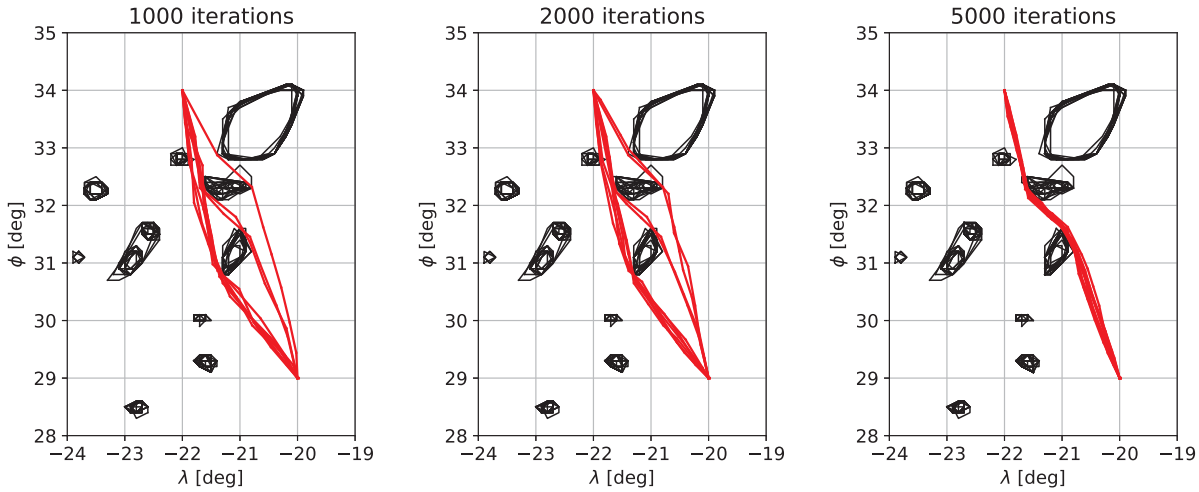


Figure 8. Relative error of the cost function with respect to the minimum flight distance as a function of the number of iterations.

V. Conclusions

In this work, we present a parallelized version of our SB-RRT*, a sampling-based algorithm for aircraft trajectory planning that considers uncertainies described by ensemble-based products. Our previous work concluded that the most time consuming step was the individual safety check required each time the graph tried to grow towards a new random sample. This function is implemented in parallel using general computing techniques on GPUs. We test the update and compare its performance to the initial version considering the same conditions. The results show that the individual safety checks now require a computational time 3 orders of magnitude lower and the entire simulation reduces its running time from days to seconds.

In virtue of the above, it would be possible to use the algorithm in near-real time operations. The algorithm could be integrated in an electronic flight bag that, based on the weather data provided by a ground station, calculates possible diversions to avoid storms. As an additional step, the safe trajectory would need to be translated into particular actions that a pilot could achieve, e.g., a change in heading.



(a) 1000 iterations. (b) 2000 iterations. (c) 5000 iterations

Figure 9. Sensitivity of the results to the maximum number of iterations.

Note that the main disadvantage of the SB-RRT* is the asymptotic optimality and its convergence rate; it approaches the solution in an inefficient manner. During the whole simulation, the algorithm is trying to minimize the cost towards each node, even though we are only interested in the trajectory between origin and goal. In consequence, many samples and iterations are wasted in the exploration of areas that are not relevant. This can be addressed through informed techniques that reduce the search space and focus the exploration on the regions in which is more likely to find the optimum.

Acknowledgment

This work has received funding from (1) the Spanish Government (Project RTI2018-098471-B-C32) and (2) the SESAR Joint Undertaking under the European Union's Horizon 2020 research and innovation programme under grant agreement No 783287. The opinions expressed herein reflect the authors' view only. Under no circumstances shall the SESAR Joint Undertaking be responsible for any use that may be made of the information contained herein. The authors would like to thank Juan Simarro, from the Spanish Met Office -AEMET- for providing us with the images shown in Figure 2.

References

[1] Eurocontrol, "Performance review report. An assessment of air traffic management in europe during the calendar year 2019," Tech. Rep., 2020.

[2] N. Dadkhah and B. Mettler, "Survey of motion planning literature in the presence of uncertainty: Considerations for UAV guidance," *Journal of Intelligent and Robotic Systems*, vol. 65, pp. 233–246, 01 2012. [Online]. Available: https://doi.org/10.1007/s10846-011-9642-9

[3] D. Hentzen, M. Kamgarpour, M. Soler, and D. González-Arribas, "On maximizing safety in stochastic aircraft trajectory planning with uncertain thunderstorm development," *Aersopace Sciencce and Technology*, vol. 79, pp. 543–553, 2018. [Online]. Available: https://doi.org/10.1016/j.ast.2018.06.006

[4] D. González-Arribas, M. Soler, M. Sanjurjo-Rivo, M. Kamgarpour, and J. Simarro, "Robust aircraft trajectory planning under uncertain convective environments with optimal control and rapidly developing thunderstorms," *Aerospace Science and Technology*, vol. 89, pp. 445–459, 2019. [Online]. Available: https://doi.org/10.1016/j.ast.2019.03.051

[5] L. He, A. Zhao, X. Wang, Z. Zhang, P. Wang, and R. Wu, "Path planning method for general aviation under hazardous weather using heuristic algorithm," in *2019 5th International Conference on Transportation Information and Safety (ICTIS)*, 2019. [Online]. Available: https://doi.org/10.1109/ICTIS.2019.8883714

[6] S. Chaimatanan, D. Delahaye, and M. Mongeau, "Hybrid metaheuristic for air traffic management with uncertainty," *Recent Developments in Metaheuristics*, pp. 219–251, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-58253-5_14

[7] W. Dai, J. Zhang, D. Delahaye, and X. Sun, "A heuristic algorithm for aircraft 4D trajectory optimization based on bezier curve," in *ATM 2019, 13th USA/Europe Air Traffic Management Research and Development Seminar*, ser. ATM Seminar 2019, 2019. [Online]. Available: https://hal-enac.archives-ouvertes.fr/hal-02178439

[8] E. Andres, M. Kamgarpour, M. Soler, M. Sanjurjo-Rivo, and D. Gonzalez-Arribas, "RRT*-based algorithm for trajectory planning considering probabilistic weather forecasts," in *The 6th ENRI International Workshop on ATM/CNS*, 2019.

[9] P. Bauer, A. Thorpe, and G. Brunet, "The quiet revolution of numerical weather prediction," *Nature*, vol. 525, pp. 47–55, 2015. [Online]. Available: https://doi.org/10.1038/nature14956

[10] C. Forster, A. Ritter, S. Gemsa, A. Tafferner, and D. Stich, "Satellite-based real-time thunderstorm nowcasting for strategic flight planning en route," *Journal of Air Transportation*, vol. 24, no. 4, pp. 113–124, 2016.

[11] R. W. Lunnon, T. Hauf, T. Gerz, and P. Josse, "FLYSAFE meteorological hazard nowcasting driven by the needs of the pilot," in *American Meteorological Society*, September 2009. [Online]. Available: https:/ams.confex.com/ams/pdfpapers/103462.pdf

[12] C. Kessinger, G. Blackburn, N. Rehak, A. Ritter, K. Milczewski, K. Sievers, and D. Wolf, "Demonstration of a convective weather product into the flight deck," in *American Meteorological Society*, February 2015. [Online]. Available: https://ams.confex.com/ams/95Annual/webprogram/Paper269015.html

[13] K. Solovey, L. Janson, E. Schmerling, E. Frazzoli, and M. Pavone, "Revisiting the asymptotic optimality of RRT*," 2020. [Online]. Available: https://arxiv.org/abs/1909.09688v2

[14] *CUDA Programming Guide*, NVIDIA Corporation, 2010.