A High-Performance Computing GNSS-aware Path Planning Algorithm for Safe Urban Flight Operations

Julian Gutierrez*
NASA Langley Research Center, Hampton, Virginia, 23666

Natasha Neogi†
NASA Langley Research Center, Hampton, Virginia, 23666

David Kaeli‡
Northeastern University, Boston, Massachusetts, 02115

Evan Dill§
NASA Langley Research Center, Hampton, Virginia, 23666

The emergence and development of advanced technologies and vehicle types have created a growing demand for new forms of flight operations. These new and increasingly complex operational paradigms, such as Advanced and Urban Air Mobility (AAM/UAM), present regulatory authorities and the aviation community with several design-and-implementation challenges – particularly for highly autonomous vehicles. An overarching and daunting task is to develop protocols that can integrate these operations without compromising safety or disrupting traditional airspace operations. A shift toward a more predictive, autonomous, risk mitigation capability becomes critical to meet this challenge. This paper proposes and evaluates a computationally-efficient path planning approach to perform pre-flight planning and autonomous in-flight re-routing to minimize exposures to selected hazards. In our evaluation, hazards associated with degraded and missing critical GPS navigation data are considered.

In this paper, we first present a high-performance computing path planning approach based on an adapted Bellman-Ford algorithm, developed in the CUDA programming language. Using the adapted path planning algorithm, we test this algorithm when encountering issues with GPS quality, and deliver an implementation that can produce flight paths that minimize exposure to risks, while maintaining a low computational burden. In our evaluation, the computation of periodic and aperiodic path updates are evaluated, prioritizing specific events as triggers for updates, based on changes to satellite availability. These critical events can lead to significant exposure to navigational hazards if not dealt with correctly.

I. Introduction

As the future of aviation adopts new and increasingly complex operational models, vehicle types, and technologies to broaden airspace capabilities and efficiency, maintaining a safe system will require recognition and timely mitigation of safety issues as they emerge, before significant consequences can occur. With prediction frameworks, we can anticipate when risky scenarios may manifest and adaptively decrease the exposure to these hazards from a flight operational perspective. One such hazard for current and future Urban Air Mobility (UAM) systems is the loss of critical navigation data due to high dependence on Global Navigation Satellite Systems (GNSS) used for reliable and continuous positioning. The effectiveness of GNSS technology is strongly dependent on direct reception of signals from at least four satellites, and the range of possible positions of those satellites. Sources of error and environmental considerations for GNSS systems have been widely explored [1, 2]. An unobstructed view of the sky is desirable to ensure accurate positioning from these systems. In situations where physical structures (e.g., buildings, terrain, etc.) visibly occlude portions of the sky, direct line-of-sight (LOS) to multiple, spatially disparate, satellites may be difficult to obtain [3].

*Research Engineer, Safety-Critical Avionics Systems Branch
†Research Engineer, Safety-Critical Avionics Systems Branch
‡Distinguished Professor, Electrical and Computer Engineering Department
§Research Engineer, Safety-Critical Avionics Systems Branch

This is a common scenario in low-altitude flight operations (e.g., UAM) and can result in position estimates that are degraded, or in some cases unavailable. Degradation or loss of this navigation data can result in catastrophic results for an autonomous air vehicle. Given the importance of minimizing exposure to hazards associated with the degradation or loss of GNSS data for UAM operations, a predictive algorithm is developed to predict the quality of GNSS systems in local airspace volumes.

Our work presented here explores the benefits of a path planning algorithm that considers GPS quality as a 4-dimensional function of space and time, and attempts to minimize flight actions in areas of poor quality. Additionally, with this algorithm's deployment on Graphics Processing Units (GPUs), we can provide safe paths to a mission planner in a matter of seconds, providing multi-kilometer trajectory data and deliver guarantees on the timeliness of results. GPUs provide massive parallelism by leveraging thousands of compute cores and high memory bandwidth. Given the inherent parallelism in Bellman Ford's approach [4], GPUs are an attractive solution to achieving real-time performance for this application.

This paper is composed of 6 sections. Section II describes the GPS simulation framework used as the main source of data for the proposed algorithm. Section III explains the important aspects of path planning for safe operations, and a detailed description of the proposed algorithm. Section IV defines the assumptions made for the simulation environment. Section V provides results on the accumulated risk across four algorithm implementations using different update intervals and the impact of event-driven updates on the accumulated risk. Finally, we summarize our findings in Section VI.

II. GPS Prediction Framework

Prior work by Dill et al. [5] showcased a predictive GPS performance monitor to support both flight planning and in-flight contingency management. This tool can help reduce navigational safety risks for low-altitude flight operations. The tool leverages real-time satellite state information (e.g., ephemeris) and a digital surface model (DSM) (i.e., a terrain model) of the operational area, to create a discretized time-varying 3D volume that computes the estimated quality of GPS-based position solutions. This framework can be run for a chosen flight environment and for a given time period. We use this framework to provide the input data for our path planning application.

A. Risk Factor

The risk factor used as input to the proposed algorithm is based on the computed output as described by Dill et al. [5]. For any voxel $v[i][j][k]$ in a 3D volume V, the position error distribution matrix is given by the following equation:

$$\mathbf{Cov}\left(\mathbf{i},\mathbf{j},\mathbf{k}\right) = \left(\mathbf{H^T H}\right)^{-1} \mathbf{H^T cov}\left(\mathbf{D}\rho\right) \mathbf{H} \left(\mathbf{H^T H}\right)^{-1} \tag{1}$$

Where $\mathbf{cov}\left(\mathbf{D}\rho\right)$ is a diagonal matrix of User Ranging Errors (URE) values, corresponding to the Space Vehicles (SVs) (i.e., satellites) in LOS to the given position $v[i][j][k]$ within the volume. URE represents the errors associated with the signal characteristics, orbit estimation errors and receiver characteristics [5]. The H-matrix represents the unitary vectors, pointing to each SV in LOS from the point $v[i][j][k]$. $\mathbf{Cov}\left(\mathbf{i},\mathbf{j},\mathbf{k}\right)$ is a 4×4 matrix representing the error distribution along each dimension. This equation provides an accurate description of the positional error distribution given by the constellation of satellites within LOS, and presents the error propagation as a mathematical effect of navigation satellite geometry on the positional measurement precision. The error propagation is computed using the matrix $Q = \left(H^T H\right)^{-1}$, describing the dilution of precision. We only consider the horizontal components of the matrix (i.e., the 2-D value) to create a risk metric that can range from 0 to 1. The risk factor metric is computed using the following equation:

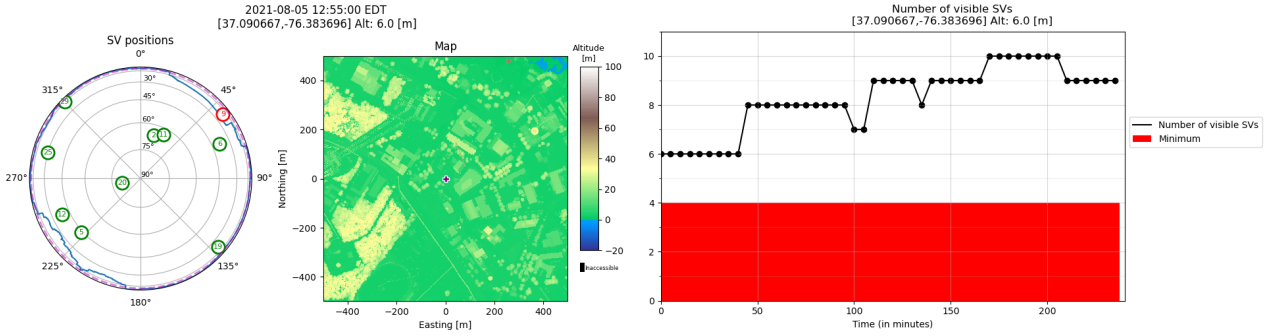
$$\mathbf{E_R} = 1 - 1/\sqrt{\mathbf{Cov}\left(\mathbf{i},\mathbf{j},\mathbf{k}\right)\left[\mathbf{0}\right]\left[\mathbf{0}\right] + \mathbf{Cov}\left(\mathbf{i},\mathbf{j},\mathbf{k}\right)\left[\mathbf{1}\right]\left[\mathbf{1}\right]} \tag{2}$$

A $\mathbf{E_R}$ of 0 means no risk is associated with GNSS hazards, as the DOP and URE values provide an accurate measurement of the position of the receiver. Equation 2 can potentially provide a negative number if the DOP and URE values are smaller than one. In these cases, $\mathbf{E_R}$ is limited to a value of zero, providing a lower bound for this metric. The following list provides a set of special cases for the value of $\mathbf{E_R}$:

- A $\mathbf{E_R}$ of 1 is used to represent inaccessible voxels. This can be a result of a building in that location or voxels marked as stay-out regions.

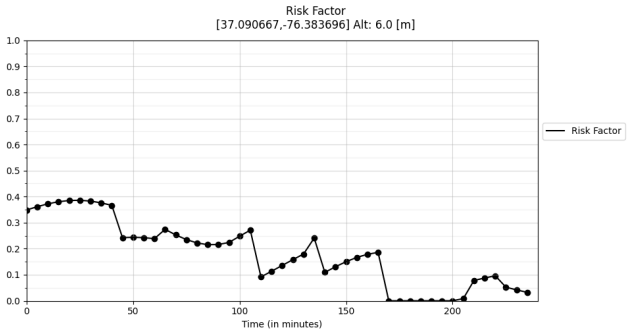
- A $\mathbf{E_R}$ of 0.99 is used to signifies accessible voxels without a sufficient number of satellites in LOS, necessary to be able to compute **Cov** $(\mathbf{i}, \mathbf{j}, \mathbf{k})$.
- A $\mathbf{E_R}$ of 0 is used for voxels with more than 15 SVs in LOS. This simplification removes the need to perform extensive matrix operations to compute an accurate measurement given the high count of available satellites in LOS.

The risk factor varies with the URE values associated with each satellite, the values of which can be obtained or estimated in a variety of ways. Additionally, individual satellite URE estimates can be updated to provide a more robust and accurate description of the position error. GPS performance is evaluated by the National Satellite Test Bed (NSTB) team and described in their GPS Standard Positioning Service (SPS) performance report [6] which provides an average estimate for URE values across all the GPS satellites. These values can also be updated in real-time from ground stations, or can be estimated by anyone with knowledge of standard GPS data analysis practices, familiarity with the relevant signal specifications, access to a GPS data archive, and can sometimes provide superior accuracy results. For example, the government commits to broadcasting the GPS signal in space with a daily global average URE of less than 2.0 m, with a 95% probability across all healthy satellites. However, the URE values are on average much better. Figure 1 shows the variation in the risk factor in an open field space in Langley, Hampton, VA, using different URE estimates for the satellites across a 12 hour period.

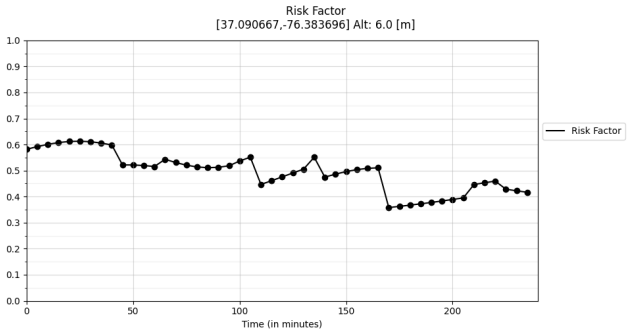


(a) Sky-plot and Map

(b) Number of Satellites in LOS



(c) URE: 1-σ



(d) URE: 2-σ

Fig. 1 URE impact on the computed risk factor, assuming individual URE assignment to each satellite in LOS (c) and (d). These observations were taken at Langley Research Center during a 12 hour period with high visibility of the sky as shown in (a).

Figure 1 (b) depicts the number of satellites in LOS at the point of interest across the 12 hours of simulation. The results in Figure 1 (c) are generated using individually estimated URE values for each satellite at a one-sigma confidence level ($\sim$ 66%), which equates to an error of approximately 1 m. Similarly, a two-sigma confidence level ($\sim$ 95%) URE is used in Figure 1 (d), resulting in 2-3 m error. Figure 1 shows that a higher URE value results in a higher risk factor, albeit it with more confidence. For the simulation environment used in this work, we found that the 1 m URE values used in the SPS report [6] were sufficient for our study.

1. Time impact

The input risk volume is smooth and monotonic, though the assumption is made that the satellites remain in LOS. Satellites leaving or entering LOS represent a non-linear step in the risk factor. Therefore, the risk factor $\mathbf{E_R}$ can be described as a function of time as follows:

$$\mathbf{E_R} = f(t) + \delta(t - t_0) \tag{3}$$

Where $f(t)$ is a linear function that depends on the satellites in LOS, as well as their geometric position in space and their respective URE values. As the satellites move through the sky, the $\mathbf{E_R}$ also changes value. The function $\delta(t - t_0)$ represents the sudden change in $\mathbf{E_R}$ as a result of a satellite moving in or out of LOS. Due to the integration of this δ function in our computations, the value of $\mathbf{E_R}$ is non-linear, which forces us to have to make certain assumptions that we discuss in more detail in section III.D and III.E.

III. 3D Path Planning

Path planning is an influential and sought-after problem to solve in graphs. Solutions such as A*, Dijkstra, and Bellman-Ford [4, 7, 8] are all well-known algorithms proven to work effectively to solve the problem of finding the shortest path between two nodes in a graph. Given the breadth of prior work in the field, a design choice was made to build upon existing techniques rather than develop a new algorithm.

Given the operating environment is a 3D elevation map, a classical flight path planning task involves finding a path from a starting point to an end point while trying to optimize select criteria (i.e., minimum distance, risk, or time) and while avoiding obstacles. Our work tries to minimize the accumulated GPS positioning error while traversing a path from the start-point to the end-point. This goal can be achieved by minimizing the accumulated risk of traversing the path. Furthermore, the path planning algorithm should also consider other factors, such as the shortest distance traveled, as a potential optimization.

Path planning is a subset of the problem known as Single Source Shortest Path (SSSP) in graph theory [9]. One of the most widely used techniques to address this problem is the A* search algorithm, which is commonly deployed as a "best-first" search in artificial intelligence, guided by a heuristic function [10]. Zhou et al. [11] describes an approach which focuses on the parallelization of A*, and acknowledges the complexity of such an endeavour. On the contrary, prior efforts by Mohammad et al. [12] conclude that other path planning techniques, specifically Bellman-Ford, present more opportunities to parallelize on a GPU and can receive a substantial speedup if tuned correctly. Hajela et al. [13] improved the performance of Bellman-Ford by using a cooperative CPU/GPU approach. Based on those promising results, our solution is also built on Bellman-Ford.

A. Path Planning to Increase Safety of Operations

Path planning to increase the safety of operations is not a new idea. Many scientists have explored this concept. Zhang and Hsu [14] worked on an approach to simulate GPS positioning errors targeting multi-path issues. Their path planning approach simplified the A* algorithm by dividing the path into 1) vertical take-off, 2) horizontal fly, and 3) vertical landing, segments. This approach proved advantageous from a computational efficiency standpoint as motion was essentially constrained to 2 dimensions for each given phase of flight. However, this approach also limited the system's capability as more advanced 3D maneuvers are not evaluated. Delamer et al. [15] proposed a similar approach to using a 2D surface at a given height to predict the safest path. Due to the computational power of GPUs and algorithm adaptations to maximize parallel computations, the techniques presented in this paper were capable of analyzing 3D path planning for the entirety of a flight plan. Analyzing a single 2D surface can provide significant performance improvements as shown in [14] and [15], but the proposed solution will not be optimal from a 3D perspective. For example, when an event occurs and a satellite leaves LOS during the path traversal, the 2D slice used for the traversal may no longer be optimal because a slice at a higher altitude may provide a reduction in the accumulated risk across the selected path. Causa et al. [16] proposed the use of additional UAVs in LOS of UAVs in critical areas to help reduce the risk of flight operations in those scenarios. However, these ideas are not explored in our work.

B. Algorithm Input

The input to the main algorithm is a 3D volume of the risk factor discussed in section II.A. This volume is composed of 1x1x1 m voxels (i.e., uniformly sized 3D grid cubes), each with a value representing the risk factor $\mathbf{E_R}$. Analogous to the Single Source Shortest Path problem in graph theory, a graph could represent the 3D volume of the risk factor.

Each voxel is a node, and each edge between nodes represents the connection to neighboring voxels. A 3D volume of voxels is used to represent the input in a structured manner, as opposed to graph data which is stored in a non-uniform fashion. Storing the data as a 3D array clearly defines the location of each voxel within the memory address space which provides a significant performance benefit in terms of memory accesses. This representation also allows the analysis of a neighboring voxel without resorting to a matrix or table, resulting in substantially better spatial locality and memory usage by the algorithm.

When analyzing the movement from one voxel to another, the risk factor associate with each voxel is weighted by the size of the movement to reach the neighboring voxel. As an example, motion along a single axis results in the risk factor being multiplied by 1, whereas motion in the diagonal connection along all three axes will result in a risk factor multiplication of $\sqrt{3}$. This increase will ensure no bias is introduced for diagonal movements in cases where such movement results in lower accumulated risk despite the extra distance traveled.

C. Bellman-Ford Adaptation

The Bellman-Ford algorithm was adapted in this effort to support the concept of minimum accumulated risk and distance, with risk as the primary objective. Thus, if there is a path with a lower accumulated risk, it will be selected as the target path. If two paths provide the same accumulated risk, the shortest path will be considered the new target path.

When the risk factor of a voxel is zero, any move from a neighboring voxel to this one will incur no added risk penalty. A zero risk value can potentially cause a problem, as moving infinitely between two neighboring voxels with zero risk would still be an optimal path from a cumulative risk perspective. To avoid an infinite loop, a comparison is done between the distance traveled to the zero risk voxel through the neighboring voxel, with the previous distance observed at the zero risk voxel. If the path through the neighbor results in a shorter distance, the neighbor is updated to reflect the shortest path. The accumulated risk remains the same for the neighbor and the zero risk voxel in this analysis.

D. Time Incorporation

As stated in the previous section, $\mathbf{E_R}$ varies with time, given the fluctuations in the satellites' positions in the sky. For this reason, we need to consider incorporating time within the developed model to ensure the final path considers changes to satellite positions as the path is traversed. For this reason, we define a **minimum update interval (MUI)**. MUI defines the minimum desirable interval for updating the risk factor volume provided to the algorithm. For this work, MUI is quantified in seconds, and will indicate the minimum amount of time that passes between "checkpoints", and an update to the risk volume is needed. Path propagation is ceased when a temporal checkpoint is reached, triggering a re-computation of the satellite locations and an update of the input risk volume. The path propagation is then resumed through the volume using the new risk volume. The following sub-section III.E describes the implementation of the time-dependent GPS-aware Bellman-Ford algorithm.

Additionally, specific trigger events that result in large changes to the risk volume are essential to consider. Given the context of this effort, we define an event as the time a satellite crosses the mask angle boundary. The mask angle is an elevation angle (commonly 10 degrees) at which satellites below this threshold are not considered in the position solution for various reasons (e.g., increased atmospheric interference). Satellites entering LOS of a receiver typically decrease the overall risk of the volume. Conversely, when satellites leave LOS, this generally increases the risk of the volume. For this reason, computations are performed for the duration of an operation to identify time instants at which satellites cross the mask angle boundary and leave LOS. After each of these events, data is refreshed to assure the most representative satellite configuration is used in computations. Recomputation is essential, as the matrix used to store the projections of the satellites across the map has to be updated to account for the new satellite configurations.

E. Final Algorithm

Three stages comprise the complete algorithm. The main stage described in algorithm 1 relates to the timing implementation described in the previous sub-section. The inputs to the algorithm are the start-point, end-point, MUI and the vehicle's speed. For simplicity, the vehicle is assumed to be moving at a constant speed. This assumption allows for a better comparison between different algorithm implementations. Additionally, it allows for a more isolated analysis of the most accurate way to utilize the 3D risk factor volume. Future work will consider a realistic representation of the vehicle's capabilities. Line 2 shows the initialization of $step_c$, an indicator of the current step being analyzed (indicating the current distance travelled at each iteration). Lines 4, 5, and 6 from algorithm 1 describe the functions provided by the GPS simulation framework, and are described in more detail in Dill et al. [5]. $step_n$ is computed, indicating the

distance needed to travel to reach the next checkpoint. Finally, the **find path** function is executed, which returns the full path once the algorithm is complete.

Algorithm 1 Main.

input
start-point
end-point
MUI (in seconds)
speed (in meters/second)
output
$path_{full}$

1: **procedure** MAIN
2: $P_S \leftarrow$ start-point
3: $P_E \leftarrow$ end-point
4: $step_c \leftarrow 0$ (in meters)
5: **while** $path_{full}$ is $None$ **do**
6: update $SV_{locations}$
7: compute the SV altitude visibility across the map
8: compute DOP and Risk metrics for the map
9: $step_n \leftarrow step_c + MUI * speed$
10: $path_{full} \leftarrow$**find_path**$(P_E, P_S, step_c, step_n)$
11: $step_c \leftarrow step_n$
12: **end while**
13: **return**: $path_{full}$
14: **end procedure**

Algorithm 2 describes the function used to initialize the algorithm and data for the GPU implementation. As stated before, it will only return the full path once the algorithm is complete. This algorithm starts by computing the **risk** volume used as the primary input for this stage in the procedure. If the current step is the first, all arrays are allocated and initialized in GPU memory. The required arrays are the following:

- **risk** array: Array used to store the risk factor values for the volume. This array is initialized to the values provided by the simulation framework.
- **ar** array: Array used to store the accumulated risk for the edge relaxation of the algorithm. We use two arrays to store the data to ensure lock-step execution of all GPU threads when updating the arrays.
- **ad** array: Array used to store the accumulated distance for the second goal of the algorithm. It also uses two arrays to ensure lock-step execution.
- **pn** array: Array used to indicate which is the previous neighbor for each voxel. This array is used to trace back the final path once the end-point is reached and no updates occur.

These arrays are initialized to infinity for all voxels except the start-point, which is initialized to zero. Following the initialization stage, a while loop continues updating the data until no additional updates occur. After this loop, a check statement confirms the completion of the algorithm. This confirmation happens by verifying whether any updates happened during the first iteration of the loop. If no updates occurred, the algorithm is done. Therefore, we can retrieve the results from the GPU and copy them to the CPU to retrieve the final path.

Finally, the relax edge function, described in algorithm 3, defines the implementation used to propagate the path through the volume. For all of the voxels in the volume, each neighbor is checked and the distance to each neighbor, represented by the quantity ddf, is computed. If the step taken $step_t$ from the neighbor is within the range given by $step_c$ and $step_n$, the accumulated risk of the neighbor is checked. In the event that the accumulated risk of the neighbor, plus the current risk of the voxel adjusted by ddf, is less than the current accumulated risk of the voxel, the **pn** array is updated to indicate the neighbor as the shortest path and the arrays are adjusted accordingly. If the accumulated risk is the same, the distance taken is evaluated to determine if it is shorter. Lines 6-12 of algorithm 3 describe these steps. Each iteration of this function represents one while loop of the find path function.

Algorithm 2 Find Path.

input
$P_S \leftarrow$ start-point
$P_E \leftarrow$ end-point
$step_c$ (in meters)
$step_n$ (in meters)
output
$path_{full}$

procedure FIND PATH
  compute **risk** volume
  if $step_c = 0$ **then**
    allocate **ar**, **ad**, and **pn** arrays on the GPU
    initialize **ar**, **ad**, and **pn** arrays
  end if
  while updates still happen, **do**
    relax edges (**risk**, **ar**, **ad**, **pn**, $step_c$, $step_n$)
  end while
  if no updates happened on the first loop **then**
    copy **pn** array back to the CPU
    retrieve the final path on the CPU
    return: $path_{full}$
  else
    return: $None$
  end if
end procedure

Algorithm 3 Relax Edges.

input
arrays (3D volumes)
$step_c$ (in meters)
$step_n$ (in meters)

procedure RELAX EDGES
  for All voxels **i** in the volume, **do**
    for each neighbor **n** of **i**, **do**
      $ddf \leftarrow \|\mathbf{n}, \mathbf{i}\|$
      $step_t \leftarrow ad[\mathbf{n}] + ddf$
      if $step_c <= step_t < step_n$ **then**
        if $(ar[\mathbf{i}] > ar[\mathbf{n}] + risk[\mathbf{i}] \times ddf)$ **then**
          $pn[\mathbf{i}] \leftarrow \mathbf{n}$
          $ar[\mathbf{i}] \leftarrow ar[\mathbf{n}] + risk[\mathbf{i}] \times ddf)$
          $ad[\mathbf{i}] \leftarrow step_t$
        else
          if $(ar[\mathbf{i}] == ar[\mathbf{n}] + risk[\mathbf{i}] \times ddf) \& (ad[\mathbf{i}] > step_t)$ **then**
            $pn[\mathbf{i}] \leftarrow \mathbf{n}$
            $ar[\mathbf{i}] \leftarrow ar[\mathbf{n}] + risk[\mathbf{i}] \times ddf)$
            $ad[\mathbf{i}] \leftarrow step_t$
          end if
        end if
      end if
    end for
  end for
end procedure

F. Implementation Details

By applying multiple CUDA optimizations, we achieved significant improvement in the execution time of the algorithm. The receding horizon propagation of the relax edges function requires multiple memory accesses when each GPU thread reads the data from each neighboring voxel within the 3D volume. This requirement presents an opportunity to leverage memory reuse. Shared memory on a GPU allows the programmer to fine-tune the data stored in the L1 cache, and better manage memory reuse. The shared memory propagation technique presented by Gutierrez et al. [17] was implemented to improve memory reuse. Additionally, this technique coalesces multiple single propagation steps into a single transaction, requiring fewer steps to converge the algorithm, and resulting in, on average, a better than 2x improvement in performance.

One of the main optimizations for Bellman-Ford is halting the algorithm once convergence has occurred. Convergence is quickly determined by checking whether updates have occurred after each step in the edge relaxation routine. We use dynamic parallelism [18] to perform this check on the GPU. The CPU launches a parent kernel, with a single thread in charge of spawning the child kernels that run the edge relaxation routines. This parent thread decides whether more iterations are needed based on the updates done by the child threads. Additionally, this arrangement of threads avoids transferring data between the CPU and GPU after each iteration, improving the readability and execution of the code.

All main arrays are stored as 3D surfaces in the GPU's main memory. The surface memory interface provided by CUDA improves spatial locality for 3D accesses that a *cudaMalloc* memory allocation cannot provide. Two accumulated risk arrays are required to allow shared memory optimizations and to prevent unsynchronized updates from propagating. This second array is also necessary for the accumulated distance array.

The neighbors for each voxel are identified using three index arrays (one per each axis). Each array indicates the relative index of the neighbor, with regards to the current voxel. These arrays are stored in constant memory, and are therefore accessed through the GPU's constant cache, reserving the bandwidth to regular caches for other operations. All of the GPU threads within a block execute the code together, resulting in each thread requesting the same array indexes. These requests are merged into a single memory read transaction and broadcast to all of threads in the block. The distance for each allowed movement, ddf, is also precomputed and stored in constant memory. Only one constant memory access is required, which avoids having each GPU thread repeat this computationally-expensive square root operation.

IV. Setup

We use an Nvidia GTX 1080 GPU in this study. The programming framework deployed for the implementation of the algorithms for computing the receding horizon propagation was CUDA 10.1 [18]. The programming language chosen for the timing logic of the algorithm was Python. Furthermore, the output data was summarized and displayed using Jupyter notebooks [19].

A. Assumptions

Considering the 3D volume described in the previous section, a risk factor of 0.0 indicated perfect satellite geometry or more than 15 SVs in the LOS. We defined the *User Ranging Error* (URE) for all satellites to be +/- 1.0m. This value allowed the risk factor to have a full range of values, allowing corner cases (i.e., zero-risk expansions) throughout the volume. Additionally, 26 movements were allowed for the vehicle in the analysis. These include all cardinal directions (6), as well as all diagonal movements (20). Diagonal movements are penalized by increasing the risk factor by the diagonal distance taken, thus avoiding diagonal preferences when propagating the receding horizon. The implementation supports limiting the scope of allowed movements to 6 directions (cardinals), resulting in an increase in the path length and a reduction in execution time due to fewer iterations through the neighbors *for* loop, reducing additional checks and code complexity.

A constant velocity of 5 m/s was selected to showcase the importance of timing in more extended simulations. As the previous section defines, events can become critical to identifying an optimal solution. With a relatively low constant speed, we can showcase the effects of these events on prolonged paths.

Finally, the scenario used in this work is of a flying emergency vehicle traveling between Massachusetts General Hospital and an intersection in the Financial District in Boston, MA. The Euclidean distance between the start-point and the end-point is 1248.8 m. This section of Boston was selected due to the height of the buildings in close proximity, which highlights the importance of simulating GPS signal blockage for proper vehicle operations. The time-frame for this simulation was March 31st, 2021, at 7 pm, starting at the hospital, and 8:30 pm starting at the intersection. The path beginning at the hospital is referenced as path P_H, and the path leaving from the intersection is referenced as path P_I.

B. Data Summary

The execution of each algorithm is summarized by collecting the distance traveled, the accumulated risk along the path, and the execution time taken to run each variation. This summary function is implemented independently from any algorithm. It verifies the selected path by traversing it point by point and computing the risk factor, accumulated risk factor, and distance independently. For each step taken, the current simulation time is used to update the satellite's position, thus the 3D volume can be updated to retrieve the most precise risk value at a point along the path. The update interval for this summary function was set to 1 s.

V. Results

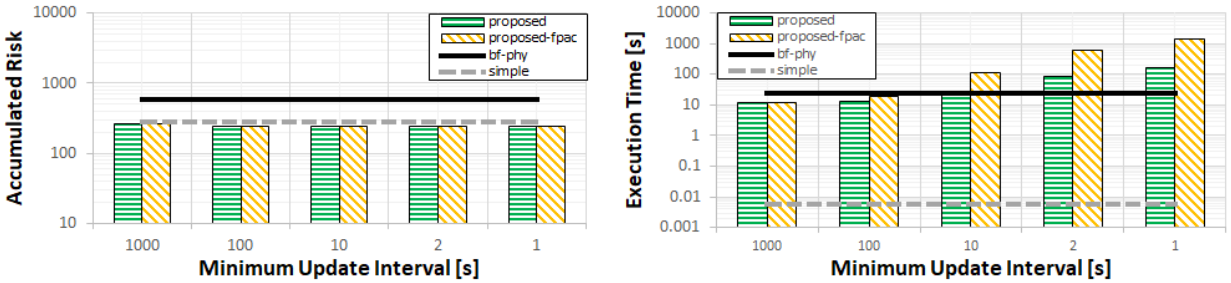
Next, we discuss the performance of our algorithms when run using different configurations. The execution time is compared to the accumulated risk measured across the path, as described in the previous section. The following implementations are evaluated in the analysis:

- **proposed**: a GPU implementation of our proposed algorithm as described in the previous section.
- **proposed-fpac**: a GPU implementation of our proposed algorithm, where the full path is computed at each checkpoint, instead of propagating the path traveled until the next checkpoint is reached.
- **bf-phy**: a GPU implementation of the Bellman-Ford algorithm that only considers the distance traveled; hence, it finds the shortest path from the start-point to the end-point.
- **simple**: a CPU implementation that moves upwards to a defined height and then traverses in the direction of the end-point until the x,y of the end-point is reached, and then it moves downward towards the end-point.

A. Accumulated Risk and Execution Time Analysis

In this test, event-driven checkpointing was not enabled to avoid biasing the results. This feature is evaluated in the following section. The key metric in the evaluation of the algorithm is the accumulated risk measured across the defined path. Figure 2 shows the accumulated risk and execution time for the various algorithms using different minimum update intervals for path P_H. As this figure suggests, the accumulated risk for the proposed and proposed-fpac algorithms converge to the same value when the MUI is decreased. Both versions achieved lower accumulated risk compared to bf-phy and the simple algorithm. This behavior was also observed in the results for path P_I, shown in figure 3.

Due to the different start times for the simulation of path P_H and P_I, the resulting accumulated risk varies. These results differed based on having more satellites for computing P_I. Despite this issue, multiple common patterns can be observed. Decreasing the minimum update interval shows, for both proposed and proposed-fpac, a decrease in the accumulated risk is achieved. Both proposed and proposed-fpac can achieve lower accumulated risk than the simple algorithm when the MUI is 10 s or lower. Additionally, there is a performance trade-off between lowering the MUI to achieve a lower accumulated risk and the execution time observed in figure 2 (b) and figure 3 (b). An MUI of 1 s translates to a proposed-fpac execution time of over 1,300 s, and over 250 s for the proposed execution time. Bellman-Ford, using only the physical distance, also proves to be a fast implementation, but with a high accumulated risk as compared to the other algorithms.



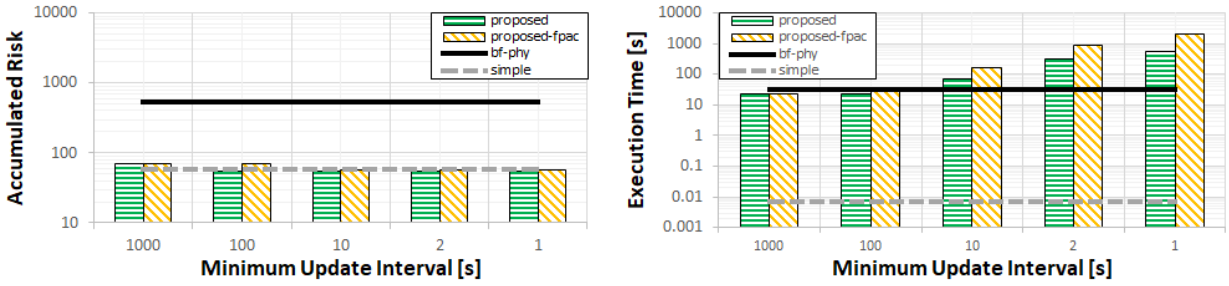
(a) accumulated risk

(b) execution time

Fig. 2 Accumulated risk and execution times for the proposed algorithms, using different minimum update intervals for path P_H.

Reducing MUI does not always result in a lower accumulated risk. While a lower MUI produces a more accurate

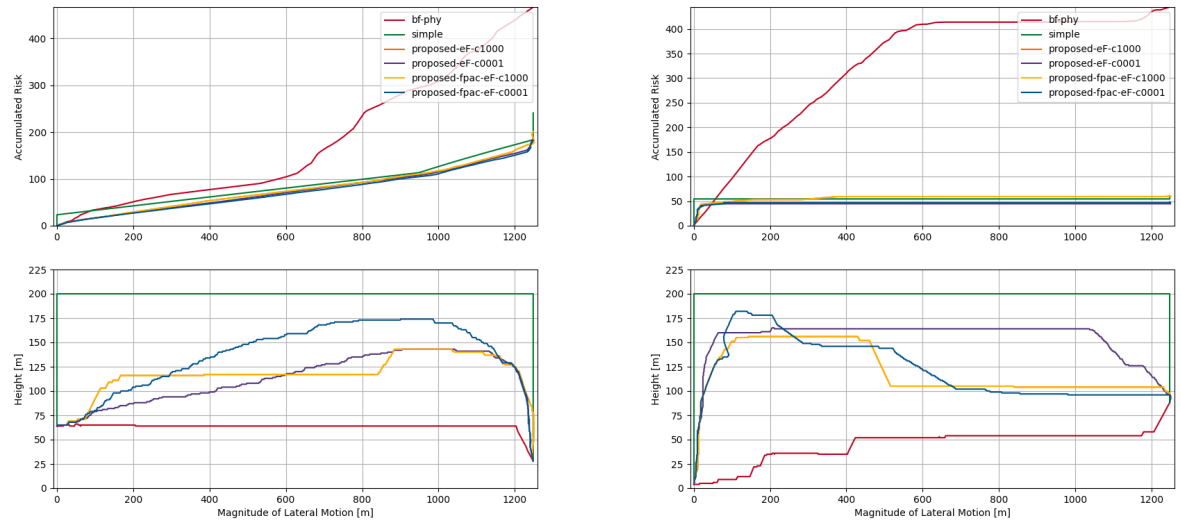
representation of the satellite locations, certain situations can cause the edge relaxation step to replace a path that would have eventually been optimal. This corner case happens when the projection of the satellites causes slower paths to replace faster paths, even though the slower path might reach a moment where the satellite constellation degrades the risk volume.



(a) accumulated risk (b) execution time

Fig. 3 Accumulated risk and execution times for the proposed algorithms, using different minimum update intervals for path P_I.

Finally, this accumulated risk metric can be used as a threshold to define whether a path is safe enough to traverse. The simple algorithm proves to be relatively computationally inexpensive and intuitive to implement. However, consistently lower accumulated risk can be achieved with the proposed solution. This difference results from the start and end points being in close proximity to taller buildings. Depending on the constellation of satellites, some of these buildings cast shadows that would affect the risk value observed by the vehicle. These scenarios could result in a safer path to move away from the building first, and then move upwards to minimize the risk posed by spending travel time under the shadow of the building. Figure 4 displays this behavior in the proposed and proposed-fpac algorithms near the end-point in (a) and the start-point in (b). The top figure shows the accumulated risk versus the magnitude of lateral motion, while the bottom plots depict the altitude of the path versus the magnitude of lateral motion.

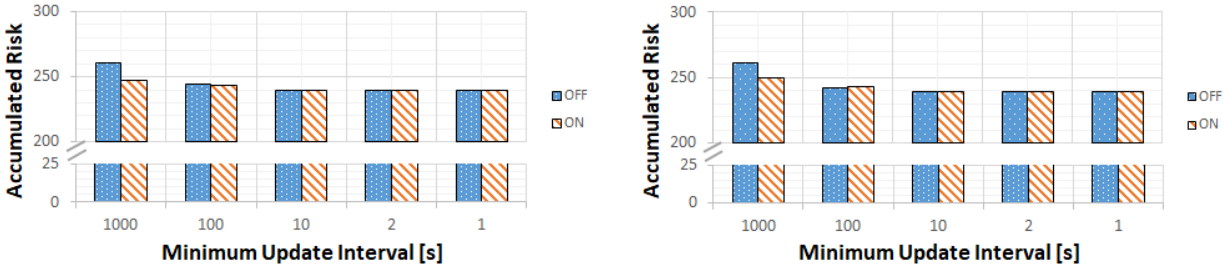


(a) (b)

Fig. 4 Accumulated risk and height of the path taken by each algorithm, for path P_H (a) and path P_I (b).

B. Event driven

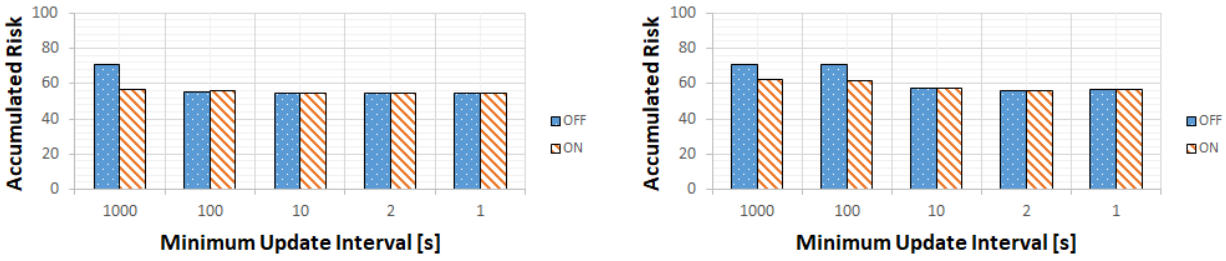
The intent of the next set of tests was to measure the impact on the accumulated risk when we consider updates triggered by events, characterized by a change in the number of available satellites. There was at least one event happening within the time scope for each path. There was an event at 43.2 s for path P_I and two events at 218.4 s for path P_H. The event for path P_I was a new satellite moving above the mask angle, while the events for path P_H were a satellite that leaves the LOS at approximately 215 s. This triggers events in the simulation seconds after the event takes place, in order to ensure we update the risk volume after the event and not when it happens, as this could result in an update with stale satellite information, rather than the new constellation.



(a) proposed (b) proposed fpac

Fig. 5 Accumulated risk comparison using event-driven analysis for the proposed algorithms, using different minimum update rates for path P_H.

Both proposed algorithms significantly reduced the accumulated risk for an MUI of 100 s or higher, when event-driven is activated. This result shows that, if the user requires a fast result, having a high MUI to reduce the execution time can still accurately map the risk, if event checkpoints are considered for path re-computations. Event-driven operation with an MUI of 1000 s * was compared to the results of the "best path" generated by using a 1 s MUI. The accumulated risk of the proposed algorithm was approximately 3.4% higher than the best path and 4.6% higher for the proposed-fpac algorithm for path P_H. For path P_I, the proposed algorithm with event-driven operation resulted in a 5.0% higher accumulated risk than the best path, and proposed-fpac, again operating event-driven, was 10.8% higher. While the overall accumulated risk was higher than the best path in these instances, the execution time was drastically reduced. The proposed and proposed-fpac algorithms executed 33.1 and 54.46 times faster, respectively, than the test used to generate the best path for path P_I. Moreover, the proposed and proposed-fpac algorithms executed 13.25x and 118x faster, respectively, than the best path for path P_H.



(a) proposed (b) proposed fpac

Fig. 6 Accumulated risk comparison using event-driven analysis for the proposed algorithms, using different minimum update rates for path P_I.

Given the scope of this work and the chosen vehicle speed, combining event-driven updates with an MUI of 10 s or faster has minimal impact. For this reason, we conclude that event-driven updates are only applicable if the MUI is 100 s or slower. Event-driven checkpoints with faster MUIs will have negligible effects.

*No updates occur in the risk volume due to MUI, because the path takes less than 1000 s to traverse.

VI. Conclusions

In this paper we analyze the performance of new 3D path planning algorithms. Our analysis considers how to minimize risk when traveling under GPS signal constraints. The proposed algorithm variations were found to attenuate the risk related to the effects of the GPS satellite movements in time, so long as the time it takes for the vehicle to move from point to point can be defined. Decreasing the minimum update interval results in a lower accumulated risk, while still achieving a reasonable execution time. The execution time of the proposed-fpac version of the algorithm increases when short MUIs are incorporated. At 1-2 s MUIs, the execution time suffers drastically, which becomes infeasible for use during in-flight analysis, although it can still be considered for pre-flight analysis. Figure 7 displays the two best solutions found for path P_H and P_I.

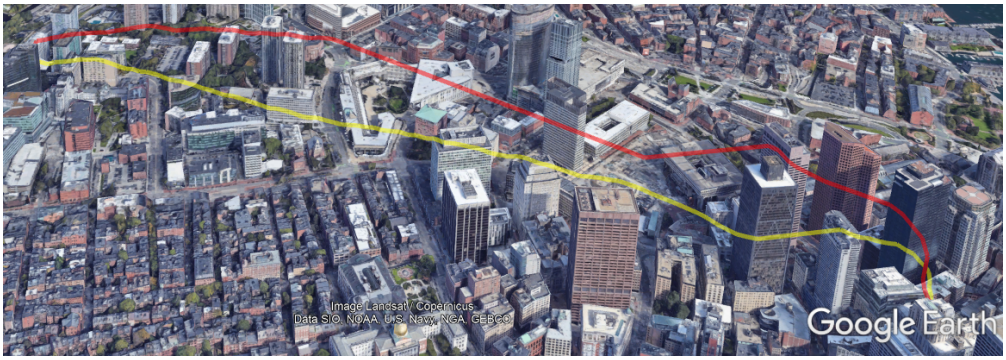


Fig. 7 Google Earth image of the safest paths for path P_H (yellow) and P_I (red).

Considering performance together with accurate GPS satellite positioning is paramount. In these scenarios, the poorer performance observed when using a slower MUI can be mitigated by incorporating event-driven updates. By targeting scenarios where satellites enter or leave the airspace, the effects in the final accumulated risk can be minimized. As an additional benefit, the proposed trajectory planning algorithms inherently avoid collisions with terrain present in the DSM. The GNSS risk factor is contingent on the LOS signals, and as obstacles limit the sky's visibility, the safest path will, in turn, be away from these obstacles.

The main goal of this paper was to produce a pre-flight algorithm that can be used to determine the waypoints along a flight path that maximize the likelihood of availability of quality navigation data from a GNSS perspective. This framework can also be repurposed as an in-flight analysis tool. Given the High-Performance Computing improvements discussed in section III, flight path segments can be rerun as needed to evaluate the current logistics and adjust the path accordingly.

However, this method does have drawbacks. The high computational load for both the GPS prediction tool and the path planning phase present a complex problem for in-flight implementation due to the dependency on high performance hardware, such as GPUs. Additionally, a very simplistic aerodynamic model of the vehicle was considered in this paper. These drawbacks will be explored in future work.

References

[1] Amin, M. G., Closas, P., Broumandan, A., and Volakis, J. L., "Vulnerabilities, threats, and authentication in satellite-based navigation systems [scanning the issue]," *Proceedings of the IEEE*, Vol. 104, No. 6, 2016, pp. 1169–1173.

[2] Pelc-Mieczkowska, R., Tomaszewski, D., and Bednarczyk, M., "GNSS obstacle mapping as a data preprocessing tool for positioning in a multipath environment," *Measurement Science and Technology*, Vol. 31, No. 1, 2019, p. 015017.

[3] Morton, Y. J., van Diggelen, F., Spilker Jr, J. J., Parkinson, B. W., Lo, S., and Gao, G., *Position, Navigation, and Timing Technologies in the 21st Century, Volumes 1 and 2: Integrated Satellite Navigation, Sensor Systems, and Civil Applications, Set*, John Wiley & Sons, 2020.

[4] Bellman, R., "On a routing problem," *Quarterly of applied mathematics*, Vol. 16, No. 1, 1958, pp. 87–90.

[5] Dill, E., Gutierrez, J., Young, S., Moore, A., Scholz, A., Bates, E., Schmitt, K., and Doughty, J., "A Predictive GNSS Performance Monitor for Autonomous Air Vehicles in Urban Environments," *Proceedings of the 34th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2021)*, 2021, pp. 125–137.

[6] Satellite Navigation Branch, A.-E. N. T. T., "Global positioning system (gps) standard positioning service (sps) performance analysis report," , 2020.

[7] Hart, P., Nilsson, N., and Raphael, B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, 1968, pp. 100–107. https://doi.org/10.1109/tssc.1968.300136, URL https://doi.org/10.1109/tssc.1968.300136.

[8] Dijkstra, E. W., "A note on two problems in connexion with graphs," *Numerische mathematik*, Vol. 1, No. 1, 1959, pp. 269–271.

[9] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., "Network flows," 1988.

[10] Hart, P. E., Nilsson, N. J., and Raphael, B., "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, 1968, pp. 100–107.

[11] Zhou, Y., and Zeng, J., "Massively parallel A* search on a GPU," *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29, 2015.

[12] Mohammad, A., and Garg, V., "Comparative analysis of floyd warshall and dijkstras algorithm using opencl," *International Journal of Computer Applications*, Vol. 128, No. 17, 2015.

[13] Hajela, G., and Pandey, M., "A Fine Tuned Hybrid Implementation for Solving Shortest Path Problems Using Bellman Ford," *International Journal of Computer Applications*, Vol. 99, No. 2, 2014, pp. 29–33.

[14] Zhang, G., and Hsu, L.-T., "A new path planning algorithm using a GNSS localization error map for UAVs in an urban area," *Journal of Intelligent & Robotic Systems*, Vol. 94, No. 1, 2019, pp. 219–235.

[15] Delamer, J.-A., Watanabe, Y., and Chanel, C. P., "Safe path planning for UAV urban operation under GNSS signal occlusion risk," *Robotics and Autonomous Systems*, Vol. 142, 2021, p. 103800.

[16] Causa, F., Fasano, G., and Grassi, M., "GNSS-aware path planning for UAV swarm in complex environments," *2019 IEEE 5th International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, IEEE, 2019, pp. 661–666.

[17] Gutierrez, J., Nina-Paravecino, F., and Kaeli, D., "A fast level-set segmentation algorithm for image processing designed for parallel architectures," *2016 6th Workshop on Irregular Applications: Architecture and Algorithms (IA3)*, IEEE, 2016, pp. 66–69.

[18] NVIDIA Corporation, "NVIDIA CUDA C Programming Guide," , 2021. Version 11.2.

[19] Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., and Willing, C., "Jupyter Notebooks – a publishing format for reproducible computational workflows," *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, edited by F. Loizides and B. Schmidt, IOS Press, 2016, pp. 87 – 90.