

Parallel rapidly exploring random tree method for unmanned aerial vehicles autopilot development using graphics processing unit processing

Lesia Mochurad¹, Monika Davidekova², Stergios-Aristoteles Mitoulis³

¹Department of Artificial Intelligence, Lviv Polytechnic National University, Lviv, Ukraine

²Faculty of Management, Comenius University Bratislava, Bratislava, Slovak Republic

³Department of Civil Engineering, School of Engineering, University of Birmingham, Birmingham, United Kingdom

Article Info

Article history:

Received Mar 20, 2024

Revised Aug 1, 2024

Accepted Aug 30, 2024

Keywords:

Acceleration

Compute unified device
architecture technology

Single instruction multiple data

System of autopilot

Unmanned aerial vehicle

ABSTRACT

Autonomous air movement systems hold great potential for transforming various industries, making their development essential. Autopilot design involves advanced technologies like artificial intelligence, machine learning, and big data. This paper focuses on developing a parallel rapidly-exploring random tree (RRT) algorithm using compute unified device architecture (CUDA) technology for efficient processing on graphics processing units (GPUs). The study evaluates the algorithm's performance in automated trajectory planning for unmanned aerial vehicles (UAVs). Numerical experiments show that the parallel algorithm outperforms the sequential central processing unit (CPU)-based version, especially as task complexity and state space dimensions increase. In scenarios with numerous obstacles, the parallel algorithm maintains stable performance, making it well-suited for various applications. Comparisons with CPU-based methods highlight the advantages of GPU use, particularly in terms of speed and efficiency. Additionally, the performance of two GPU models, NVIDIA RTX 2070 and T4 is compared, with the T4 demonstrating superior performance for similar tasks. Future research should explore integrating multiple algorithms for a more comprehensive UAV autopilot system. The proposed approach stands out for its stability and practical applicability in real-world autopilot implementations.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Lesia Mochurad

Department of Artificial Intelligence, Lviv Polytechnic National University

Kniazia Romana str., 5, Lviv 79905, Ukraine

Email: lesia.i.mochurad@lpnu.ua

1. INTRODUCTION

In the 21st century, all possible technologies are developing towards autonomy and automation, as noted in [1], [2]. After the success of developing a complete autopiloting system for ground vehicles (for example, Tesla's autopilot: see [3]), the demand for technologies for automatic route planning for all types of vehicles moving in space has rapidly increased. Vehicles that can move independently, fulfilling their tasks, have become one of the main directions of development of modern science and technology [4]–[6]. In recent years, unmanned aerial vehicles (UAVs) have received special attention, as the airspace does not suffer from traffic problems and contains much fewer obstacles to movement, which allows them to move much faster and safer, performing a wide range of tasks.

In our work, the main focus is on the development of a system that will ensure the autonomous movement of vehicles in airspace (autopiloting), since even now UAVs are used or integrated into all areas of human activity, for example:

- Agricultural and agrarian activities – drones are used for watering or sowing a field and detecting pests, and diseases among cultivated plants [7].
- Mapping and cartography – creating, updating, and detailing current maps by collecting information about the environment under study through aerial images [8].
- Civilian and research aerial photography – obtaining unique footage while avoiding environmental pollution and any interference with the ecosystem under study [9], [10].
- Transportation—UAVs are used to transport materials and medicines in places where the availability of roads and transportation is limited or in cases where there is a need to achieve high speed of transportation [11], [12].

Today, the market value of autopilot technology for UAVs is growing rapidly and has positive growth forecasts for the next 15 years [13], [14], which demonstrates the developers' efforts to create a better offer, as well as the corresponding need for investors to have such technology. Delivery services and the transportation sector as a whole, despite their constant development, are limited by two significant drawbacks: the speed and the need for human presence to perform transportation tasks, which causes a large number of complications, namely: waiting time for order delivery, loss of profit due to the need to keep people for transportation (including the need to pay for their labor), deterioration of traffic on the roads (trucks take up a lot of space and move slowly compared to cars), creating Modern aircraft are capable of covering hundreds of kilometers in a short period of time, which is much better than ground transportation. The availability of UAVs with high-quality software as a vehicle for the transportation of small and medium-sized cargo can solve most of the above problems. Despite all the attempts and research conducted to solve the problem of transportation using UAVs, this system has not been globally established (except for a few private companies that are currently testing their systems, such as Amazon) due to the lack of public implementations of path and flight path algorithms. Similar problems exist in other areas, and their solution depends on the availability of public research on the problem of autopiloting aircraft by building their flight path, which is what we are considering in this paper. Thus, a large number of problems in various areas of human activity are solved with the help of UAVs, the creation of which requires public research on the problem of autopiloting, which is the topic of our work.

Delivery services and transportation, despite their continuous development, are limited by two significant drawbacks: speed and the need for human presence to perform transport tasks, leading to numerous complications. These include delivery waiting times, loss of revenue due to the need for personnel for transportation (including their payment), worsened traffic congestion (freight vehicles take up space and move slowly compared to passenger vehicles), road accidents, and many others [15]–[17]. Modern drones are capable of covering hundreds of kilometers in short periods, which is a much better indicator than ground transportation modes. The presence of drones with quality software as a transportation means for small to medium-sized cargo can solve most of the aforementioned problems. Despite all attempts and research conducted to address transportation issues using drones, this system has not been globally optimized (except for a few private companies currently in the testing stage of their systems, such as Amazon) due to a lack of public implementations of path planning and flight trajectory algorithms. Similar problems exist in other areas, and their resolution depends on the availability of public research on the issues of drone autopiloting, specifically in constructing flight trajectories, which is the subject of our work. Therefore, a large number of problems in various human activities are solved using drones, the creation of which requires public research on autopiloting issues, which is the topic of our work.

The majority of the analyzed literature sources share a common ideology. The importance of researching means and algorithms for ensuring the autonomous operation of aircraft to perform various types of tasks in various areas of human activity. It should be noted that none of the reviewed studies claimed to be the only "solution" to all the problems of this problem. Instead, the authors wrote that currently (as of the time of writing) there is no algorithm or method that would ideally solve the problem of trajectories and autopiloting in general for all tasks (in other words, solutions are selected for specific, localized problems), but there is a great demand for working prototypes and algorithms, which creates a certain "arms race" that continues to this day.

Feng [18] considers a path planning method for a mobile robot using a combination of a fast exploratory random tree (RRT) and a particle swarm optimizer (PSO) [19]. The main emphasis is on the speed and efficiency of the new algorithm in a cluttered environment, where successful obstacle avoidance is confirmed by the results of a computer experiment. Our work proposes a parallel RRT algorithm based on compute unified device architecture (CUDA) technology for optimal processing on a graphics processing unit (GPU). Using an improved PSO algorithm, Yang *et al.* [20] also consider a way to solve the problem of drone autopiloting by searching for a flight path. They drew attention to the need for UAVs to be able to operate in

an unknown environment (not always the entire structure, obstacles, and other environmental parameters will be known in advance, which can lead to unforeseen consequences, loss of the UAV itself, and in the case of attack UAVs, destruction of allied equipment and soldiers), and used the PSO algorithm created for this purpose, improving it with optimizers. Despite all their efforts, the authors failed to fully optimize the algorithm and its shortcomings, such as not guaranteed convergence and the possibility of fixation in a local minimum. The potential convergence of the algorithm is the main drawback that will be addressed and corrected in this paper.

Voelker *et al.* [21] developed a "lightweight" autopilot for small UAVs using the Dubin's Path algorithm, emphasizing that different tasks and types of drones require distinct algorithms and approaches. They highlighted that attack drones need complex, precise paths at high speeds, whereas regular drones benefit from navigation algorithms for unknown spaces where speed is less critical. The authors chose Dubin's Path for its simplicity and extensive documentation but acknowledged significant limitations: the system is only for small UAVs, works in two-dimensional space, ignores obstacles, requires precise control, lacks self-improvement capabilities, and supports only one UAV at a time. In our study, all these shortcomings are taken into account and corrected or improved in one way or another, since our proposed system can work with UAVs of different sizes, and is not limited in the number of dimensions (RRT [22] is one of the best algorithms that work in three-dimensional space, and its efficiency increases with the number of dimensions used). Our system can be used for a wide range of UAV tasks [23], it is also highly flexible and scalable (it can be improved by installing additional modules and algorithms), able to take into account obstacles and ensure their avoidance, does not require precise control of the course, speed, and other UAV parameters, and can work with a large number of vehicles simultaneously, which makes it better than the one discussed above.

Li and Hansen [24] describes the importance of research in the field of UAV software, since every year the number of tasks that could be simplified with the help of, for example, drones or even possible only with their help is growing rapidly, and with it the need for autonomous, but still controlled, UAV operation. The authors studied the effectiveness of several pathfinding algorithms (Dijkstra's [25], Bellman Ford's [26], Floyd-Warshall's [27], A* [28], D* [29]) and compared them, paying special attention to the A* algorithm, as its results were the best of all. Despite the good results obtained and the goal achieved (drone flight trajectories built), the study has a drawback - the environment in which the path search was performed is well known: the authors conducted a preliminary analysis of the environment and prepared it to ensure the correct operation of the algorithms (A* among them). In our study, this drawback is corrected by using another algorithm that can work in a completely unknown, new environment, which was one of the main reasons for its choice.

Jin *et al.* [30] investigate the problem of safe indoor UAV navigation using a series of algorithms and methods to achieve accurate results. The article was written recently (at the time of writing) and once again emphasizes the importance of studying the autonomous operation of UAVs to perform various tasks, the number of which is growing every year, and the need for effective algorithms (especially public ones) for building a path and flight trajectory is growing exponentially. The authors took into account the need to navigate in unfamiliar environments and decided to implement their auto-learning: a separate algorithm builds a pseudo-map of the environment using data from the UAV's cameras and sensors, and only after the first algorithm is complete does the drone start building a route. Although this approach proved to be effective, its implementation is very cumbersome, multi-layered, and requires some special equipment to work correctly, and the route is built only after a complete study of the environment, which undoubtedly takes a lot of time. Our approach to solving the tangential problem also studies the environment, but it does so simultaneously with the construction of the flight path (RRT algorithm), which, firstly, simplifies implementation, and secondly, reduces the overall time required to launch the UAV along the built route. In addition, as mentioned above, our implementation allows for future improvements that will provide good results for different tasks, unlike the approach used in this paper: a large number of interconnected algorithms will be much harder to improve, as it will guarantee changes in the implementation. It should also be noted that the probabilistic roadmaps (PRM) algorithm used by the researchers, as well as its results, strongly depend on the chosen primary strategy, which, as Alarabi *et al.* [31] notes, is difficult to choose correctly.

Xu *et al.* [32] proposed a method for planning the trajectory of mobile cable-driven parallel robots based on the developed multi-agent random tree, which allows for adaptation of the sampling in the subspace and efficiently generates possible paths. It also presents dynamic control validation, heuristic optimization, and performance metrics to ensure smooth and optimal path generation, verified through simulations and prototype experiments in complex scenarios using CoppeliaSim software. Karve and Kapadia [33] investigate a way to solve the problem of building a flight path for UAVs using the Dijkstra algorithm, focusing on achieving certain constraints, such as covering the largest possible area for video recording, thus creating a certain modification of this algorithm. The authors managed to achieve guaranteed generation of the optimal flight path (one of the main properties of the Dijkstra algorithm), but there are two common problems in their implementation: Inability to take into account and avoid obstacles in its path; Inability to work in unknown environments (the

algorithm requires a graph of the environment with weights that correspond to certain features as input. Building and preparing such a graph takes a lot of time). The RRT algorithm, which we use in our study, provides a solution to the above problems. The research in [34], [35], the authors also consider the problem of increasing the efficiency of path construction in the state space using the RRT algorithm. The authors solved this problem by parallelizing the RRT algorithm on central processing unit (CPU), which resulted in a performance acceleration of 3 [34], and 4 [35] times. In our study, we implement the parallel RRT algorithm on a GPU using CUDA technology [36], [37]. Since the GPU is much better at handling large amounts of computation than the CPU, we expect to achieve better results than those obtained by the authors of the papers mentioned above, and thus expect to achieve a better solution to the pathfinding problem. We have previously analyzed various modern technologies of distributed and preliminary computing, in particular, those covered in [38], [39].

Our work focuses on the development of a system that ensures the autonomous movement of vehicles in airspace (autopiloting), addressing the significant gap in public implementations of path and flight path algorithms. Our approach proposes a parallel RRT algorithm based on CUDA technology for optimal processing on a GPU, improving the efficiency and capability of UAV autopilot systems in unknown environments. Despite all attempts and research conducted to address transportation issues using drones, this system has not been globally optimized (except for a few private companies currently in the testing stage of their systems, such as Amazon) due to a lack of public implementations of path planning and flight trajectory algorithms. Similar problems exist in other areas, and their resolution depends on the availability of public research on the issues of drone autopiloting, specifically in constructing flight trajectories, which is the subject of our work.

We aim to develop an efficient parallel RRT algorithm for execution on GPUs, significantly enhancing automated trajectory planning for UAVs. The majority of the analyzed literature sources share a common ideology: the importance of researching means and algorithms for ensuring the autonomous operation of aircraft to perform various types of tasks in various areas of human activity. This highlights the ongoing demand for effective and publicly available solutions to improve UAV autopiloting capabilities. The main contribution of this paper is as follows:

- A new parallel RRT method is proposed and implemented using CUDA technology for optimal processing on the GPU.
- A comprehensive study and comparison of the effectiveness of parallel algorithms using GPU and CPU for automated trajectory planning of UAVs was conducted. It was found that the proposed parallel algorithm is highly efficient, in particular in conditions of increased task complexity and state space dimensionality.
- Additionally, we compared the NVIDIA RTX 2070 and T4 GPUs, revealing the advantage of the latter in performing similar tasks. This emphasizes the importance of choosing the optimal hardware to improve the performance of parallel computing.

Overall, the work makes a significant contribution to the field of automated trajectory planning for UAVs by introducing a new parallel algorithm and providing important comparative performance analyses.

2. PROBLEM STATEMENT

Let $X \in \mathbb{R}^n$ denote the state space of the unknown environment, and $O \subseteq X$ denote the subspace of the state space in which the moving object (UAV) is not in a state of collision with an obstacle (does not cross, intersect, or pass through). Let $x \in X$ denote the configuration of the flying vehicle (a set of primitive directives, and states). The algorithm takes as input the initial state x_{init} and the target state x_{goal} , such that $x_{init} \in O$ and $x_{goal} \in O$. It is necessary to find a path $P : (x_0, x_1, x_2 \dots, x_{end})$, where $x_0 = x_{init}$, $x_{end} = x_{goal}$, and P lies in the space O , avoiding all obstacles and guaranteeing safe arrival at the state x_{goal} , if such a path is possible.

3. METHOD

3.1. Overview of the sequential rapidly-exploring random tree algorithm

The RRT algorithm, originally devised for pathfinding, is a widely utilized method across various domains. The algorithm operates by constructing a stochastic tree to efficiently explore the state space. The starting position, serving as the origin of the search, acts as the root node of the tree. It follows an iterative approach wherein, during each iteration, a new point is randomly chosen from the state space. Subsequently, the nearest node within the tree, considering obstacles, is located relative to the generated point. This node is then connected to the generated point at a predefined distance, a hyperparameter of the algorithm, in the direction of the generated point. This process iterates until either the path to the desired state is discovered or the maximum allowed number of iterations, determined by another algorithm parameter, is reached.

Here's the pseudocode representing the formal definition of the RRT algorithm:

```

Function RRT(x_init, x_goal, X, O):
Initialize tree T(x_init,)
done ← false
While not done:
x_rand ← RandState(X, O)
x_near ← Nearest(x_rand, T)
x_new ← Steer(x_near, x_rand)
e_new ← {x_near, x_new}
If not CheckCollision(e_new, O):
Add vertex x_new to V: V = V ∪ {x_new}
Add edge e_new to E: E = E ∪ {e_new}
If x_new = x_goal:
Path(x_new)
done ← true
Return T

```

This pseudocode represents the RRT algorithm using functions: *RandState*(X, O) – function for generating x_{rand} ; *Nearest*(x_{rand}, T, X) – function to find the nearest node; *Steer*(x_{near}, x_{rand}) – a function for building a state from the nearest node in the x_{rand} direction; *CheckCollision*(e_{new}, O) – function to check if there is an obstacle between x_{new} and x_{near} . *Path*(x_{new}) – a function for traversing the tree T in the reverse order from x_{new} to the root x_{init} to get the path. The RRT function takes input x_{init} , x_{goal} , X , and O , and returns the tree T representing the path found by the RRT algorithm. Where $X \in \mathbb{R}^n$ – state space; $O \subseteq X$ – is the set of obstacles, which is obviously a subset of the state space; $T = (V, E)$ – tree built as a result of the RRT algorithm; V – set of tree nodes $V \subseteq X$ but $V \cap O = \emptyset$; E – set of tree edges; x_{init} – initial state; x_{goal} – target state; $x_{init}, x_{goal} \in V$; x_{rand} – randomly selected state, $x_{rand} \in X$; x_{near} – is the nearest node of the tree T to x_{rand} , $x_{near} \in V$; x_{new} – selected point based on x_{near} and x_{rand} , $x_{new} \in X$; e_{new} – edge connecting the vertices x_{near} and x_{new} .

3.2. Parallel rapidly-exploring random tree algorithm

Before describing the parallelized version of RRT, we believe it is necessary to give an asymptotic estimate of the complexity of computing sequential RRT:

$$O(N, M) = O_{rand}(N) + O_{near}(N) + O_{steer}(M * N * \log(N)) = M * N * \log(N)$$

where N is the number of states, M is the number of obstacles, $O_{rand}(N)$ – asymptotic function evaluation *RandState*; $O_{near}(N)$ – asymptotic function evaluation *Nearest*, $O_{steer}(M * N * \log(N))$ – asymptotic evaluation of functions *Steer* and *CheckCollision*.

Accordingly, to develop a parallel RRT that will run on the GPU, we have been thinking within the framework of the SIMD (Single Instruction Multiple Data) architecture, which will allow us to best utilize the hardware capabilities provided by GPUs. Accordingly, we propose to parallelize the following algorithm functions: *Nearest*, *Steer*, *CheckCollision* – they are ideal for execution on a large number of GPU processors, are autonomous, i.e., do not require synchronization, and will not exceed the execution time limit (for example, for CUDA technology, it is one second). With such parallelization, we should achieve the following asymptotic computational complexity:

$$O_P(N, M) = O_{rand}(N) + O_{near}\left(\frac{N}{P}\right) + O_{steer}(M * N * \log(N) / P) = \frac{M * N * \log(N)}{P}$$

where N is the number of states, M is the number of obstacles, and P is the number of GPU cores.

In our work, we used the following technologies to develop and optimize computing processes:

- CUDA: Software development that uses GPUs to perform general-purpose computing. With the help of CUDA, computational acceleration was achieved due to parallel processing of data by the GPU.
- Python: Using Python as an interpreted programming language for rapid prototyping, development, and solving computational problems. Python was chosen because of its wide range of tools and ease of use.
- Numba: Using Numba as a higher-level abstraction for CUDA technology to simplify development, make code more secure, and reduce potential errors. Numba provides JIT compilation for efficient GPU and CPU computing.

- NumPy: Using NumPy to process and analyze data, as well as to efficiently implement computations on arrays and matrices of large sizes. NumPy has been used to solve scientific computing and data processing problems.
- Matplotlib: Using Matplotlib to visualize data and calculation results. The library allows you to create various types of graphs for easy analysis and presentation of work results.

Thus, the combination of these technologies allowed us to develop efficient software to solve the computational problem, taking advantage of GPUs and providing a user-friendly interface for development and visualization. When implementing the algorithm, we use the "Numba" library decorator "cuda.jit", with the help of which we define functions that will be executed on a GPU that supports CUDA technology. Accordingly, the following functions were implemented,

```

– euc_distance_2d_device(x1,y1,x2,y2) – function for calculating the Euclidean distance between points on the GPU,
@cuda.jit(device=True)
def euc_distance_2d_device(x1,y1,x2,y2):
d = math.sqrt((x2-x1)**2+(y2-y1)**2)
return d
– def distanceKernel(d_out,x,y,d_V,nov): – kernel function responsible for parallel calculation of the distances between the generated point and the tree nodes,
@cuda.jit()
def distanceKernel(d_out,x,y,d_V,nov):
i = cuda.blockIdx.x*cuda.blockDim.x + cuda.threadIdx.x
if i < nov:
d_out[i] = euc_distance_2d_device(x,y,d_V[0,i],d_V[1,i])
– dArray(x,y,V,nov) – function to encapsulate the exchange of information between the host and the device when calculating distances,
def dArray(x,y,V,nov):
d_V = cuda.to_device(V) # copies the input data to a device array on the GPU
d_distance = cuda.device_array(nov) # creates an empty array to hold the output
BPG = (nov + TPB - 1)//TPB # computes number of blocks
distanceKernel[BPG,TPB](d_distance,x,y,d_V,nov)
return d_distance.copy_to_host()
– def ccKernel(d_out,x,y,d_O,all_radaii): – kernel function responsible for the parallel calculation of collisions,
@cuda.jit()
def ccKernel(d_out,x,y,d_O,all_radaii):
i = cuda.blockIdx.x*cuda.blockDim.x + cuda.threadIdx.x
#flag = 1
if i < all_radaii.size:# and flag ==1:
d_out[i] = euc_distance_2d_device(x,y,d_O[0,i],d_O[1,i])>all_radaii[i] # should be 1 for no collision
#flag = d_out[i]
– def dArray_CC(x,y,obs_coors,allowable_radaii): – a function for encapsulating the exchange of information between the host and the device when calculating collisions with obstacles,
def dArray_CC(x,y,obs_coors,allowable_radaii):
noo = allowable_radaii.size
d_all_radaii = cuda.to_device(allowable_radaii) # copies the input data to a device array on the GPU
d_O = cuda.to_device(obs_coors) # copies the input data to a device array on the GPU
d_collision = cuda.device_array(noo) # creates an empty array to hold the output
BPG = (noo + TPB - 1)//TPB # computes number of blocks
ccKernel[BPG,TPB](d_collision,x,y,d_O,d_all_radaii)
return d_collision.copy_to_host()

```

4. RESULTS AND DISCUSSION

Numerical experiments for this work were conducted on the following hardware systems:

- NVIDIA GeForce RTX 2070
- NVIDIA T4 GPU
- CPU Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz.

In this study, we do not use datasets, because the RRT algorithm solves the problem of finding a path in an arbitrary state space, so we developed a procedure to generate state spaces with an arbitrary number of obstacles using the NumPy library.

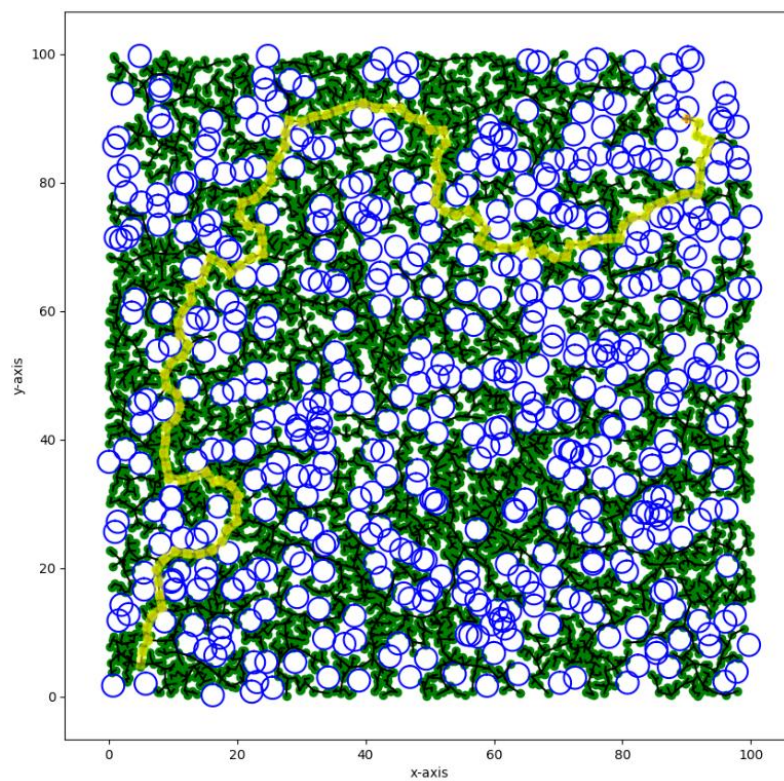
```
def generate_obstacles(obstacles_count: int, radius: float, state_dimension: int) ->
[np.ndarray, np.ndarray]:
"""
:param obstacles_count: the number of obstacles to generate
:param radius: the radius of each obstacle
:param state_dimension: the dimensionality of the state space
:return: two NumPy arrays containing the radii and coordinates of the generated obstacles
"""
obstacles_radii = radius * np.ones(obstacles_count)
obstacles_coordinates = state_dimension * np.random.rand(2, obstacles_count)
return obstacles_radii, obstacles_coordinates
generate_obstacles(512, np.sqrt(3) / 2, 100)
```

In this pseudo-code, the `generate_obstacles` function generates a specified number of obstacles with the specified radius and dimensions of the state space. The result is two NumPy arrays: one with the radii of the obstacles and the other with their coordinates in the state space. An example of the function call shows how 512 obstacles with a radius equal to $\sqrt{3}/2$ can be generated in a state space of size 100. We grouped all the environments on which we conducted the experiments and saved them in a publicly available dataset [40].

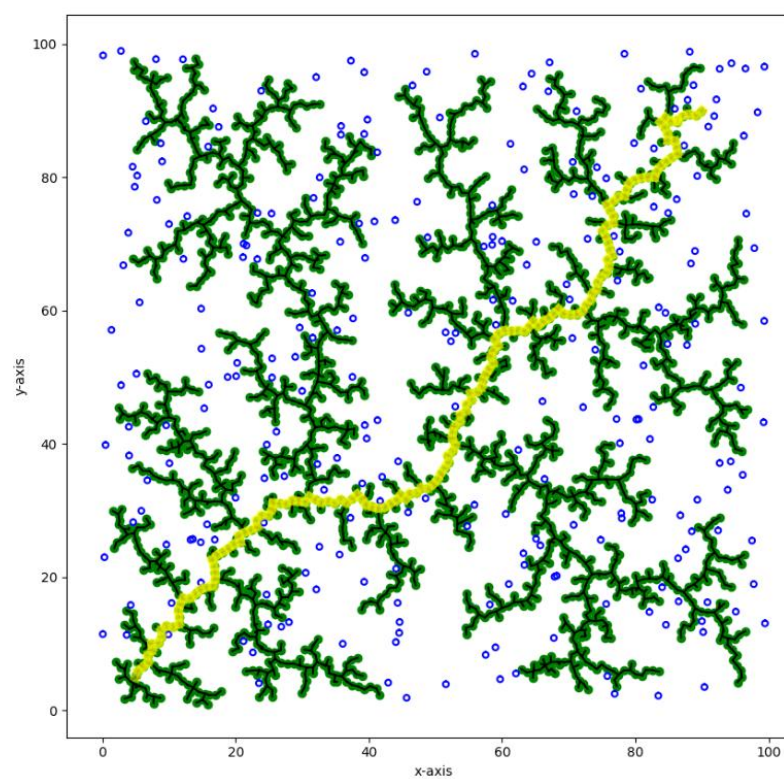
The following are examples of building a path search tree in the generated state spaces. Figure 1 shows the search tree built using the RRT algorithm (green color indicates the nodes of the tree, they are connected by black lines) and the path found (yellow line) between the points (5;5) and (90;90) in the state space 100×100 and the number of obstacles (blue color) 512; NVIDIA GeForce RTX 2070. In Figure 1(a), we see the result of our parallel RRT algorithm, which successfully finds a path between the states with coordinates (5;5) and (90;90) in 5.4438 seconds, the found path is visualized by a yellow line. The green dots connected by black lines provide a graphical representation of the search process in the 100×100 state space. In Figure 1(b), we can observe the work of the sequential RRT executed on the CPU, the execution time is 5.181 seconds. Although the environments in Figures 1(a) and 1(b) are the same size (100×100), it is easy to see that the environment in this figure is much simpler (128 obstacles in Figure 1(a) vs. 512 in Figure 1(b)). Comparing these two cases, we can see that when running parallel RRT on a video card, with a more complex environment, the algorithm explored most of the state space at almost the same time as the sequential algorithm in a much simpler space, which significantly increases the probability of finding a path in a constant time. This is the main idea behind the application of the algorithm we have developed: by parallelizing and using the hardware capabilities of GPUs, we not only speed up the solution of the problem of finding a path in the state space but also increase the probability of finding this path in some time.

The following formula was used to calculate the speedup based on the measurements of the execution time of the RRT algorithm and the parallel version we developed: $S = \frac{T_s}{T_p}$, where S is the ratio of the time T_s required to perform calculations on a single-processor computer system to the time T_p to solve the same problem on a parallel system (in our case, a GPU) using a parallel algorithm. Thus, the time measurements of the sequential and parallel algorithms for state spaces of 100×100 and 200×200 are presented in Tables 1 and 2 and visualized in Figures 2 and 3, respectively. The parallel speedup rates for different state spaces are presented in Tables 3 and 4.

Let's start analyzing the results of the numerical experiments by evaluating the speed of the two algorithms under study. We can see that the GPU-parallelized algorithm is faster than the conventional sequential algorithm. In particular, as the parametric complexity of the problem increased, the difference in execution time increased significantly, which, as expected, is due to the reduction in the overall complexity of the algorithm (including synchronization time) and the parallel use of CUDA cores for computational operations. We can see that after applying 2048 or more obstacles in the generated environment defined in the state space of dimension 100×100 , the computation time of the sequential algorithm on the CPU increases approximately twice with each step, while the GPU-parallelized algorithm remains in place (as can be seen in Table 1), against the geometric growth of the number of obstacles (and, as a result, computations). When the number of obstacles was less than 1024, we observed an advantage in the speed of the sequential algorithm on the CPU, however, the performance of parallel computations even with 512 obstacles differed in the worst case by only 0.3. In other words, even with the synchronization time, the parallelized algorithm gave good speed results, which makes it suitable for relatively lightweight problems (with a small number of obstacles in small state spaces).



(a)



(b)

Figure 1. A search tree is constructed and a path is found between the points (5;5) and (90;90) in the 100×100 state space: (a) the number of obstacles is 512 using the parallel RRT algorithm, (b) the number of obstacles is 128 using the sequential RRT algorithm on the CPU

Table 1. Time of execution of sequential and parallel RRT algorithms in a state space of size 100×100 in seconds for points (5;5) and (90;90)

Number of obstacles	Sequential (CPU)	Parallel	
		NVIDIA GeForce RTX 2070	NVIDIA T4 GPU
256	4.448984146118	7.9860494136	5.2984178066
512	8.516096115112	12.8609824180	8.9158022403
1,024	128.2285559177	85.5747628211	76.2808237076
2,048	122.1113481521	74.0722501277	66.9172148705
4,096	241.3418300151	74.8022329807	68.6118636131
8,192	468.4661860466	76.2232465744	77.3167104721

Table 2. The execution time of the sequential and parallel RRT algorithms in a 200×200 state space in seconds for points (15;20) and (185;190)

Number of obstacles	Sequential (CPU)	Parallel	
		NVIDIA GeForce RTX 2070	NVIDIA T4 GPU
256	43.2499878406	34.6423091888	15.5784752368
512	42.1336429119	31.8523607254	30.4305672646
1,024	176.3126482963	74.0990240573	74.7495670319
2,048	299.9120779037	76.4668069839	75.3771859169
4,096	555.5035376548	76.6121499538	75.6121499538
8,192	984.0232720375	76.8012492656	75.8806983947

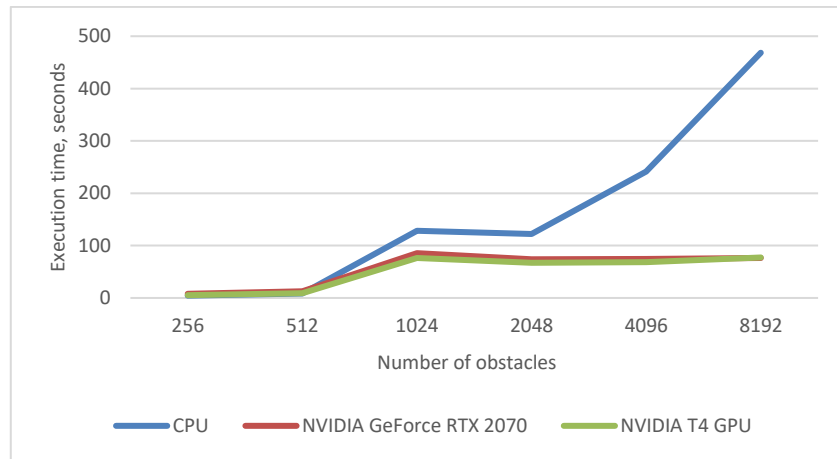


Figure 2. Running time of sequential and parallel RRT algorithms in a state space of size 100×100 for points (5;5) and (90;90)

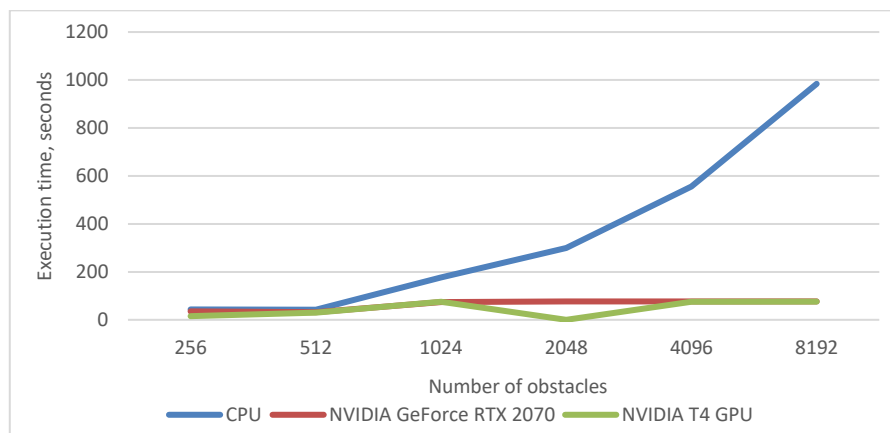


Figure 3. Time of execution of sequential and parallel RRT algorithms in a state space of size 200×200 in seconds for points (15; 20) and (185; 190)

Table 3. Parallel acceleration of the RRT algorithm based on measured data in a 100×100 state space in seconds for points (5;5) and (90;90)

Number of obstacles	NVIDIA GeForce RTX 2070	NVIDIA T4 GPU
256	0.557094	0.839682
512	0.662165	0.955169
1,024	1.498438	1.681006
2,048	1.648543	1.824812
4,096	3.226398	3.517494
8,192	6.145975	6.059055

Table 4. Parallel acceleration of the RRT algorithm based on measured data in a 200×200 state space in seconds for points (15;20) and (185;190)

Number of obstacles	NVIDIA GeForce RTX 2070	NVIDIA T4 GPU
256	1.248473	2.776266
512	1.322779	1.384583
1,024	2.379419	2.358711
2,048	3.922121	3.978818
4,096	7.250854	7.346752

Comparing our results with those of other researchers, the authors of articles [34], [35], who used CPUs for parallelization, we can say that we have achieved a better result than they did and, with a large number of obstacles, our speedup rates are much higher than those of the authors (6.145975239 (Table 2) of peak speedup versus 3.512 [34] and 4.123 [35] for a 100×100 space, the authors of the articles do not provide results for larger spaces). We also conducted an experimental comparison of two NVIDIA GPUs: RTX 2070 and T4. Both graphics cards were introduced in 2018, so we consider their comparison to be appropriate. As you can see from Figures 2 and 3 and Tables 2 and 4, the acceleration rates of the T4 are slightly higher, even though the power of the RTX 2070 is higher. This is because the T4 video card was created specifically for performing such tasks, for which it received a large amount of video memory (16 GB).

5. CONCLUSION

In conclusion, this paper addressed the problem of UAV pathfinding, focusing on obstacle avoidance and trajectory planning in unknown environments. We implemented the RRT algorithm and improved its performance through parallelization using CUDA on GPUs, resulting in significantly better efficiency compared to a CPU-based approach. The parallel algorithm successfully avoided all obstacles, confirming the effectiveness of the method. This approach enables UAVs to navigate complex spaces without requiring precise environmental data, making it suitable for challenging conditions such as forests or indoor spaces. Future research should focus on integrating additional algorithms to enhance the overall performance of UAV autopilot systems and further expand their application potential.

ACKNOWLEDGEMENTS

This work is funded by the European Union's Horizon Europe research and innovation program under grant agreement No 101138678, project ZEBAI (Innovative methodologies for the design of Zero-Emission and cost-effective Buildings enhanced by Artificial Intelligence). This work was supported by the Slovak Research and Development Agency under the contract No. APVV 19-0581.




REFERENCES

- [1] S. Chiodo, "Human autonomy, technological automation (and reverse)," *AI and Society*, vol. 37, no. 1, pp. 39–48, 2022, doi: 10.1007/s00146-021-01149-5.
- [2] J. A. J. Alsayaydeh *et al.*, "Development of programmable home security using GSM system for early prevention," *ARNP Journal of Engineering and Applied Sciences*, vol. 16, no. 1, pp. 88–97, 2021.
- [3] K. N. Maanyu, D. G. Raj, R. V. Krishna, and S. B. Choubey, "A study on Tesla autopilot," *Issue*, vol. 171, no. 171, 2020.
- [4] M. A. Al Noman *et al.*, "A computer vision-based lane detection technique using gradient threshold and hue-lightness-saturation value for an autonomous vehicle," *International Journal of Electrical and Computer Engineering*, vol. 13, no. 1, pp. 347–357, 2023, doi: 10.11591/ijece.v13i1.pp347-357.
- [5] S. Abdul-Khalil, S. Abdul-Rahman, S. Mutalib, S. I. Kamarudin, and S. S. Kamaruddin, "A review on object detection for autonomous mobile robot," *IAES International Journal of Artificial Intelligence*, vol. 12, no. 3, pp. 1033–1043, 2023, doi: 10.11591/ijai.v12.i3.pp1033-1043.
- [6] A. Jadhav, S. K. Shandilya, I. Izonin, and R. Muzyka, "Multi-step dynamic ensemble selection to estimate software effort," *Applied Artificial Intelligence*, vol. 38, no. 1, 2024, doi: 10.1080/08839514.2024.2351718.
- [7] M. P. A. F. Kiki, S. A. R. M. Ahouandjinou, K. M. Assogba, and T. Sutikno, "Bibliometric analysis and survey on electronic nose used in agriculture," *International Journal of Electrical and Computer Engineering*, vol. 14, no. 2, pp. 1369–1381, 2024, doi: 10.11591/ijece.v14i2.pp1369-1381.




- [8] M. E. Tjahjadi, F. Handoko, and S. S. Sai, "Novel image mosaicking of UAV's imagery using collinearity condition," *International Journal of Electrical and Computer Engineering*, vol. 7, no. 3, pp. 1188–1196, 2017, doi: 10.11591/ijece.v7i3.pp1188-1196.
- [9] C. J. Pinza-Jiménez and Y. A. Garcés-Gómez, "Assessing the performance of random forest regression for estimating canopy height in tropical dry forests," *International Journal of Electrical and Computer Engineering*, vol. 13, no. 6, pp. 6787–6796, 2023, doi: 10.11591/ijece.v13i6.pp6787-6796.
- [10] K. Telli *et al.*, "A comprehensive review of recent research trends on unmanned aerial vehicles (UAVs)," *Systems*, vol. 11, no. 8, 2023, doi: 10.3390/systems11080400.
- [11] B. Benmhahe, J. A. Chentoufi, and M. A. Basmassi, "A 3D reconstruction-based method using unmanned aerial vehicles for the representation and analysis of road sections," *International Journal of Electrical and Computer Engineering*, vol. 14, no. 2, pp. 1552–1564, 2024, doi: 10.11591/ijece.v14i2.pp1552-1564.
- [12] A. Saboor, S. Coene, E. Vinogradov, E. Tanghe, W. Joseph, and S. Pollin, "Elevating the future of mobility: UAV-enabled intelligent transportation systems," *arXiv-Computer Science*, pp. 1-7, 2021.
- [13] P. Antriksh, "Drone autopilots market," *Aerospace and Defense*, 2023. [Online]. Available: <https://www.kingsresearch.com/drone-autopilots-market-76>
- [14] S. A. H. Mohsan, M. A. Khan, F. Noor, I. Ullah, and M. H. Alsharif, "Towards the unmanned aerial vehicles (UAVs): a comprehensive review," *Drones*, vol. 6, no. 6, 2022, doi: 10.3390/drones6060147.
- [15] V. Garg, S. Niranjani, V. Prybutok, T. Pohlen, and D. Gligor, "Drones in last-mile delivery: a systematic review on efficiency, accessibility, and sustainability," *Transportation Research Part D: Transport and Environment*, vol. 123, 2023, doi: 10.1016/j.trd.2023.103831.
- [16] A. Houari and T. Mazri, "Optimal model of vehicular ad-hoc network assisted by unmanned aerial vehicles and information-centric networking to enhance network performance," *International Journal of Electrical and Computer Engineering*, vol. 14, no. 2, pp. 1788–1796, 2024, doi: 10.11591/ijece.v14i2.pp1788-1796.
- [17] I. Fedorchenko, A. Oliinyk, J. A. J. Alsayaydeh, A. Kharchenko, A. Stepanenko, and V. Shkarupilo, "Modified genetic algorithm to determine the location of the distribution power supply networks in the city," *ARN Journal of Engineering and Applied Sciences*, vol. 15, no. 23, pp. 2850–2867, 2020.
- [18] Z. Feng, "An optimized algorithm based on random tree and particle swarm," *International Journal of Robotics and Automation (IJRA)*, vol. 3, no. 2, pp. 107–111, 2014.
- [19] H. M. Salman, A. K. M. Al-Qurabat, and A. A. R. Finjan, "Bigradient neural network-based quantum particle swarm optimization for blind source separation," *IAES International Journal of Artificial Intelligence*, vol. 10, no. 2, pp. 355–364, 2021, doi: 10.11591/ijai.v10.i2.pp355-364.
- [20] Y. Yang, X. Xiong, and Y. Yan, "UAV formation trajectory planning algorithms: a review," *Drones*, vol. 7, no. 1, 2023, doi: 10.3390/drones7010062.
- [21] A. Voelker *et al.*, "Three-layer aircraft control: Path-planning, guidance, model predictive controller as autopilot," *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 43, no. 15, pp. 81–86, 2010, doi: 10.3182/20100906-5-jp-2022.00015.
- [22] S. Davarzani and M. T. Ejaz, "A 2D path-planning performance comparison of RRT and RRT* for unmanned ground vehicle," *IAES International Journal of Robotics and Automation*, vol. 13, no. 1, pp. 105–112, 2024, doi: 10.11591/ijra.v13i1.pp105-112.
- [23] F. S. Hutabarat, I. M. D. Sitanggang, J. A. I. Damanik, G. P. B. Knight, and A. Sagala, "Unmanned aerial vehicle design for disaster victim search and rescue operation using wireless sensor network localization," *IAES International Journal of Robotics and Automation (IJRA)*, vol. 11, no. 4, p. 288, 2022, doi: 10.11591/ijra.v11i4.pp288-303.
- [24] A. Li and M. Hansen, "Obstacle clustering and path optimization for drone routing," *9th International Conference for Research in Air Transportation*, pp. 1–8, 2020.
- [25] M. A. Khan, "A comprehensive study of dijkstra's algorithm," *SSRN Electronic Journal*, 2023, doi: 10.2139/ssrn.4559304.
- [26] J. Cho, G. Lee, and S. Lee, "An automated direction setting algorithm for a smart exit sign," *Automation in Construction*, vol. 59, pp. 139–148, 2015, doi: 10.1016/j.autcon.2015.05.004.
- [27] I. H. Toroslu, "Improving the floyd-warshall all pairs shortest paths algorithm," *arXiv-Computer Science*, pp. 1-5, 2021.
- [28] S. Alani, A. Baseel, M. M. Hamdi, and S. A. Rashid, "A hybrid technique for single-source shortest path-based on a* algorithm and ant colony optimization," *IAES International Journal of Artificial Intelligence*, vol. 9, no. 2, pp. 356–363, 2020, doi: 10.11591/ijai.v9.i2.pp356-363.
- [29] L. Leenen and A. Terlunen, "A focussed dynamic path finding algorithm with constraints," *IEEE International Conference on Adaptive Science and Technology, ICASST*, 2013, doi: 10.1109/ICASTech.2013.6707501.
- [30] Q. Jin, Q. Hu, P. Zhao, S. Wang, and M. Ai, "An improved probabilistic roadmap planning method for safe indoor flights of unmanned aerial vehicles," *Drones*, vol. 7, no. 2, 2023, doi: 10.3390/drones7020092.
- [31] S. Alarabi, C. Luo, and M. Santora, "A PRM approach to path planning with obstacle avoidance of an autonomous robot," *2022 8th International Conference on Automation, Robotics and Applications, ICARA 2022*, pp. 76–80, 2022, doi: 10.1109/ICARA55094.2022.9738559.
- [32] J. Xu, B. G. Kim, Y. Lu, and K. S. Park, "Optimal sampling-based path planning for mobile cable-driven parallel robots in highly constrained environment," *Complex and Intelligent Systems*, vol. 9, no. 6, pp. 6985–6998, 2023, doi: 10.1007/s40747-023-01114-3.
- [33] D. Karve and F. Kapadia, "Multi-UAV path planning using modified Dijkstra's algorithm," *International Journal of Computer Applications*, vol. 175, no. 28, pp. 26–33, 2020, doi: 10.5120/ijca2020920816.
- [34] D. Devaurs, T. Simeon, and J. Cortés, "Parallelizing RRT on distributed-memory architectures," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2261–2266, 2011, doi: 10.1109/ICRA.2011.5979751.
- [35] J. Ichnowski and R. Alterovitz, "Parallel sampling-based motion planning with superlinear speedup," *IEEE International Conference on Intelligent Robots and Systems*, pp. 1206–1212, 2012, doi: 10.1109/IROS.2012.6386194.
- [36] T. A. Sadoon and M. H. Ali, "Deep learning model for glioma, meningioma, and pituitary classification," *International Journal of Advances in Applied Sciences*, vol. 10, no. 1, pp. 88–98, 2021, doi: 10.11591/ijaas.v10.i1.pp88-98.
- [37] L. Mochurad, O. V. Matviiv, H. Lema, and R. Vilhutska, "CUDA-based algorithm for lidar position determination in mobile robotics," *CEUR Workshop Proceedings*, vol. 3426, pp. 193–203, 2023.
- [38] L. Mochurad, O. Kotsiumbas, and I. Protsyk, "A model for weather forecasting based on parallel calculations," *Lecture Notes on Data Engineering and Communications Technologies*, vol. 159, pp. 35–46, 2023, doi: 10.1007/978-3-031-24468-1_4.
- [39] L. Mochurad, K. Shakhovska, and S. Montenegro, "Parallel solving of fredholm integral equations of the first kind by tikhonov regularization method using openmp technology," *Advances in Intelligent Systems and Computing*, vol. 1080, pp. 25–35, 2020, doi: 10.1007/978-3-030-33695-0_3.
- [40] L. Mochurad, "Generated_obstacles," *Research Gate*, 2024. [Online]. Available: https://www.researchgate.net/publication/379083608_generated_obstacles

BIOGRAPHIES OF AUTHORS






Lesia Mochurad    received a Ph.D. in Engineering Science from Lviv Polytechnic National University, Ukraine. She is an Associate Professor of the Department of Artificial Intelligence Systems. She is the author or co-author of more than 150 scientific and scientific-methodological works, seven monography and two scientific manuals, and a textbook, has two copyright certificates for inventions. Scientific interests: computational intelligence, mathematical modeling, distributed and parallel computing technologies, ensemble learning, big data processing, abelian groups of symmetry. In addition, she is a reviewer for leading journals such as the journal of intelligent & fuzzy systems, applied sciences, aerospace, sensors, mathematics, fractal fract, agronomy, and PLOS ONE. She is also a member of the editorial board of the journal International Journal of Reconfigurable and Embedded Systems (IJRES). She can be contacted at email: lesia.i.mochurad@lpnu.ua.



Monika Dávideková, Ph.D.    is a researcher at the Faculty of Management, Comenius University Bratislava, Slovak Republic. She graduated with summa cum laude and obtained her Ph.D. degrees in the field of telecommunication and in the field of management at the Faculty of Electrical Engineering and Information Technology at Slovak University of Technology and at the Faculty of Management at Comenius University, respectively. She has been working previously in the field of software engineering, project management, validation and verification as well as organizational management in several fields (finance, water management, wind energy, and automotive). At present, her research interests and teaching activities are in management information systems, knowledge management, internet of things, virtual and augmented reality, virtual distributed teams, virtual collaboration and in business analytics. She can be contacted at email: davidekova2@uniba.sk.



Dr. Stergios-Aristoteles Mitoulis    is the Leader of MetaInfrastructure.org and www.bridgeUkraine.org. He is the Head of Structures at the University of Birmingham and Associate Professor with a focus on infrastructure safety, resilience, sustainability, and digitalisation. Delivering integrated solutions to climate hazards and multiple stressors. He holds a 5-year Diploma from Aristotle University and MSc in Sustainable design of infrastructure to natural hazards and a Ph.D. in 2008. He is CEng MICE, a member of the ASCE, EAEE, IABSE, and FHEA-UK. He has published more than 130 journal and conference papers. Stergios has worked as PI, Co-I and researcher for more than 25 research projects, including coordinating and leading a 1.7m-Euro HORIZON-MSCA-SE multipartner project on climate-aware resilience for sustainable and interdependent infrastructure systems. He can be contacted at email: s.a.mitoulis@bham.ac.uk.