Indonesian Journal of Electrical Engineering and Computer Science
Vol. 26, No. 1, April 2022, pp. 450~461
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v26.i1.pp450-461

Path planning for assisting blind people in purposeful navigation

Andrés Díaz-Toro, Paula Mosquera-Ortega, Gustavo Herrera-Silva, Sixto Campaña-Bastidas
School of Basic Sciences, Technology and Engineering, Universidad Nacional Abierta y a Distancia, Pasto, Colombia

Article Info

Article history:

Received Oct 27, 2021
Revised Feb 8, 2022
Accepted Feb 12, 2022

Keywords:

Dynamic environment
Graphics processing unit
Occupancy grid
Parallel programming
Performance evaluation
Real time processing
Replanning

ABSTRACT

Blind people present dificulties for reaching objects of interest in the daily life. In this sense, the integration of a path planning module for assisting blind people in purposeful navigation is noteworthy. In this work, we present an algorithm that leverages the high capability of an embedded computer with graphics processing unit (GPU) NVIDIA Jetson TX2 for computing optimal paths to objects of interest. The algorithm computes the optimal path to the objective, considering changes in the environment and changes in the position of the user. The algorithm is efficient for computing new paths when the environment changes by reusing parts of previous computations. In order to compare the performance, the algorithms were implemented and evaluated in MATLAB, C++ and CUDA, for different size of the grid and percentage of unknown obstacles. We found that the implementation on GPU has a speed up of 20 times W.R.T the implementation in C++ and more than 400 times W.R.T the implementation in MATLAB. These results boost us to integrate this module to our main system based on a stereo camera and a haptic belt and so provide to the user assistance in purposeful navigation at real time.

Corresponding Author:

Andrés Díaz-Toro
School of Basic Sciences, Technology and Engineering, Universidad Nacional Abierta y a Distancia
Pasto, Colombia
Email: andres.diaz@unad.edu.co

1. INTRODUCTION

People with severe visual impairment, especially the ones with total loss of sight, struggle to move and carry out activities in unfamiliar environments [1], which reduces their opportunities in education, work, health, and social activities, affecting considerably their quality of life. Purposeful navigation consists in setting a destination and computing a path to guide the user to the target. There are few systems that provide assistance to blind people in purposeful navigation, like [2]-[6]. Some reasons for this shortage are listed next. i) These applications require to process dense data at real time, so a computer with high computational capabilities is needed; ii) These applications require that the user moves and carries the hardware, so the computer must also be portable; iii) These applications require complex algorithms to localize in indoors without the help of global positioning system (GPS) (like the ones presented in [7]-[9]), detect static and dynamic obstacles, and build incrementally a representation of the environment; iv) Until few years ago, the algorithms for object detection were not efficient. With the advent in the last years of advanced algorithms of artificial intelligence applied to computer vision (like the ones presented in [10]-[12]), efficient modules for object detection than run on graphics processing unit (GPU) have been plausible; v) Most algorithms for path planning were focused on video games and driverless vehicles (like the ones presented in [13]-[16]), especially due to economic interests.

These difficulties have been overcome, due to advances in embedded computers with high capabilities, in vision sensors, and in efficient algorithms of computer vision. Moreover, more researchers are interested in developing these assistive tools and more funds are available. Next, we present outstanding systems for assisting blind people to navigate safely in unknown environments and that include a module of path planning for guiding them to objects of interest.

In the paper, Lee and Medioni [2] consists of a glass mounted RGBD camera with inertial measurement unit (IMU) sensors, a laptop carried by the user in the back that runs the software at real time, and a vest-type interface with four vibration motors. It estimates the position of the camera with visual odometry, segments the floor using normal vectors and random sample consensus (RANSAC), creates a 3D voxel map and a 2D traversable map, and guides the user to objects of interest in dynamic indoor environments. An optimal path is computed in the 2D map using D*Lite algorithm [17]. Audio and haptic feedback is delivered to the user while he/she can specify the destination using a smart-phone application.

In the paper, Islam *et at*, [3], an indoor navigation system for assisting blind people is described. The system consists of a cap with IR receivers, an Arduino Nano for processing data, a headphone for the feedback with voice commands and an ultrasonic sensor for detecting obstacles. The experiments were carried out in a room with 16 IR transmitters that allow to know the position of the user. The system loads a grid of the room that defines free and occupied cells. The user can define the destination cell through a braille keypad. After defining the destination cell, the system finds a path to it, with a simple algorithm that generates non optimal paths.

The CCNY smart cane [4] is based on the Google Tango software [18] that runs on a mobile device (the phone/tabled hybrid Lenovo Phab2 with integrated 3D depth sensor, gyroscope and accelerometers), mounted on the user's chest. It provides simultaneous localization and mapping (SLAM), has the ability to remember key visual features such as doors and rooms (stored in an area description file (ADF)) and is able to find objects by utilizing IR sensors. The system plans a path using A* search algorithm [19] and guides a visually impaired person to objects of interest in indoor environments with both haptic (two vibrating motors mounted on the white cane) and audio feedback. For an easier guidance the iterative end-point fit algorithm [20] is used, so the user receives information of the rotation before walking straight.

The wearable system [5] presents a planner that combines static and dynamic planning, that runs at real time. It consists of a binocular camera for obstacle detection, a GPS/inertial navigation system for pose estimation in outdoors, a laptop, and a headset. In global planning, a local target is determined by aiming-tracking to the global path for each frame. In local planning, an improved dynamic weighted A* algorithm [21] is implemented. Moreover, an autoregressive model predicts the position of dynamic obstacles with changing acceleration. The proposed algorithm produced a smaller number of expansion nodes than A* in both static and dynamic environments.

In the paper, Zhang and Ye [6], a robotic navigation aid for blind people is presented. It estimates its pose using a RGB-D camera (Intel RealSense D435) and an IMU. Visual-inertial odometry is carried out with color, depth and inertial data for computing the pose of the user in indoor environments, at real time. The floor plane is computed from the depth data and IMU data. A graph of nodes is built, where each node is a point of interest and the edges represent the distance between the nodes. The system uses a motorized rolling tip mounted on a white cane in order to guide the user along the planned path (robocane mode). Other option for feedback is through a speech interface (white-cane mode). The mode is automatically selected using a decision tree model. The software was developed based on ROS framework. The A* algorithm was used for path planning, in similar way to [22].

In this paper, we present the selection, implementation and evaluation of a path planning algorithm (efficient dynamic search (EDS)), implemented in MATLAB, C++ and CUDA, running on a laptop and on an embedded computer (the NVIDIA Jetson TX2 developer kit), for guiding at real time a blind person in indoor/outdoor environments. This algorithm is robust to dynamic obstacles and to changes in the pose of the user since he/she moves following the computed path. We evaluated the path planning algorithm in an occupancy 2D grid that was built incrementally in real environments with data from a stereo camera, the Zed Mini. Synthetic data was included to the grid for defining unknown obstacles and up-sample was carried out for obtaining grids of different sizes. The contributions of our work are next; i) The design of a system for assisting blind people in purposeful navigation, considering the requirements for a real application; ii) An algorithm for path planning implemented on GPU for real time performance, robust to changes in the environment and to changes in the position of the user; iii) The experimental results that validate the efficiency of the proposed algorithm.

This paper is organized as follows. In section 2, we present the methodology used for selecting and implementing the path planning algorithm that best fits to the requirements of our project. In section 3, we describe the experiments carried out for computing an optimal path in an occupancy grid with real and simulated data, with different size, percentage of unknown obstacles, and executed in different programming languages and platforms. Finally, in section 4, we present the main conclusions obtained in this work.

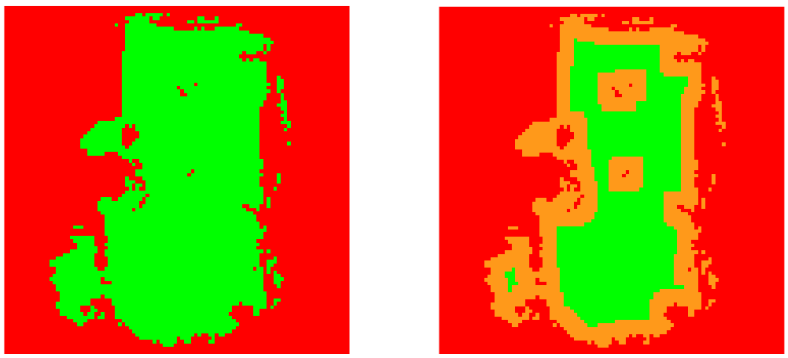
2. RESEARCH METHOD

In the first stage of our project, we achieved to build a global occupancy grid that differentiates free and occupied regions. The vibration patterns generated by a belt, warn to the user that an obstacle is close and its direction, in order to avoid it. The camera is the Zed Mini from StereoLabs and the embedded platform is the Jetson TX2 from NVIDIA as shown in Figure 1(a). The belt was built with four vibration motors, which are controlled with an Arduino Nano. We carried out experiments in a paved courtyard with size of 9m long and 5m width. It is surrounded by walls, windows and small gardens. Boxes are used as obstacles as shown in Figure 1(b). For more details, see [23].

Currently, the system builds an occupancy grid with a processing speed of 7,56fps. The occupancy grid results of projecting 3D points to the floor and defining a threshold in height. Depth data was obtained from depth maps of the stereo camera. This grid is a global representation of the environment since local information is merged during the whole trajectory, considering the pose of the user, delivered by the camera. We use one of these grids, computed in the paved courtyard during three minutes as shown in Figure 2(a). The position of the user is defined by a cell (cell-size user). However, the real volume of the person must be considered for path planning. For this purpose, we dilate the grid in order to increase the size of the obstacles. The resulting grid is shown in Figure 2(b).

(a) (b)

Figure 1. Wearable system and space for experiments, (a) blindfolded person wearing the vision-based system, with the camera in the chest and (b) paved courtyard with boxes as obstacles



(a) (b)

Figure 2. Management of the volume of the user into the occupancy grid, (a) occupancy grid of 100×100 cells of the space shown in Figure 1(b), the size of each squared cell is 10cm long and (b) occupancy grid after applying dilation, in order to deal with the volume of the user

In the second stage of our project, we propose to define targets, it means, destination cells where objects of interest are located. Once, the destination cells are defined, a path planning algorithm must estimate an optimal trajectory from the current position of the user to the selected destination cell. Next, the system guides the user to the target with vibration patterns generated by the belt. This functionality is known as purposeful navigation. Figure 3(a) shows the modules that make up the main system. The global map makes possible to execute a path planning algorithm that estimates an optimal path over a much larger space than in the case of local representations. The small size of the cells allows to represent detailed information

of the environment and therefore to make correct computation of the optimal path. However, it presents a higher computation load in planning. Moreover, the movement of the user is considered to be without non-holonomic constraints. It is turned into eight neighbors in the rectangular grid with costs of 10 units in straight direction and 14 units in diagonal direction.

The user follows an estimated path. If a discrepancy between the map and the environment is detected then the map is updated, a new path from the current location to the goal is computed and this process is repeated until the goal is reached as shown in Figure 3(b).

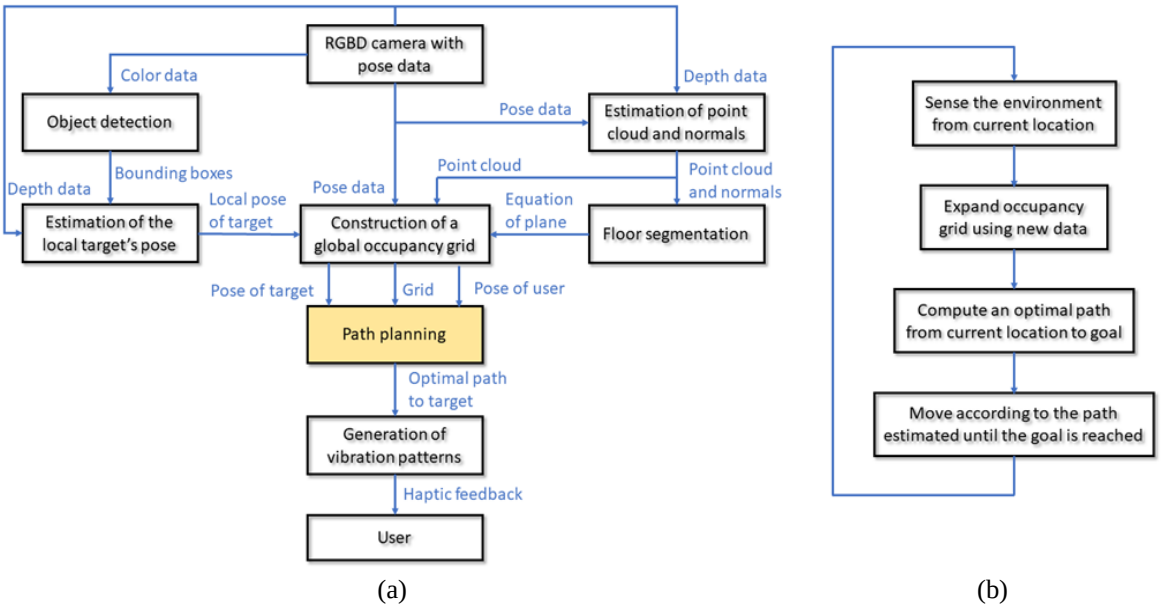


Figure 3. Diagrams for describing the main system and the path planning process, (a) modules that make up the main system, the path planning module is highlighted in yellow and (b) processes to reach the goal

In real situations, it is common that when the user is moving to a goal through an estimated path, he/she realizes that the path is blocked, for example, due to a door that was closed. Conversely, a shortcut that was not in the previous map could exist. In that moment, the path must be computed again at real time to keep moving.

2.1. Context and requirements of our application

For selecting the most appropriate algorithm for path planning we consider the next context.

- The environment is a plain terrain with free and occupied spaces that are represented by cells so the cost to traverse these cells can take two values: a cost to straight neighbors of 10 units and 14 units for diagonal neighbors (non-uniform cost). The cost to occupied cells is infinity.
- The environment can change suddenly, for example, a door can be opened or closed, a person can move and obstruct the path of the user, a chair can be moved to another position, among others (dynamic environment). Moreover, the representation of the environment is built incrementally along the time due to the limited range of sensing (range defined by the horizontal field of view of the camera of 80° and the range of measurement in depth of 3.0m) and it includes data with uncertainty (partially known environment), so a discrepancy between the map and the environment could occur.
- The user is moving from the starting cell in order to reach the goal cell, considering the haptic instructions and the computed path (moving user).
- The user needs permanent and immediate feedback for reaching the goal and this feedback must result from updated representations of the environment, considering that it can change suddenly (real time processing and feedback)
- The destination (static goal) must be defined as coordinates in the 2D occupancy grid like in [24]. We propose, as future work, to compute it by averaging the 3D points obtained from depth data inside a bounding box generated by a module of object detection like [10]-[12].

2.2. Comparison of path planning algorithms

A planner generates a sequence of actions or a sequence of states. A path planner computes a sequence of state transitions through a graph, creating a path from point A to point B. We are interested in a path planner that generates an optimal path in a dynamic environment. Based on the context and requirements of the application previously explained, we make a comparison of some outstanding algorithms for path planning.

Breadth first search [25] computes an optimal path in static environments for every node of the graph, which has uniform cost. It uses a first in, first out (FIFO) queue: nodes are removed from the front and placed at the back. As result, the exploration is equal in all directions. This continuous expansion of unvisited nodes closest to the start is called the wavefront optimality principle.

Dijkstra's algorithm [26] is used in graphs of non-uniform costs (varying costs). It favors the exploration of lower cost paths by using a priority queue based on the total cost from the start location to the current location. In this way, the wavefront optimality principle is preserved in non-uniform cost graphs, resulting that the exploration is faster in nodes with lower cost. It computes the path to all nodes of the graph, like Bread First Search.

A* algorithm [19] is optimized for computing a path for a single location, unlike the previous algorithms that expands in all directions. It is done by favoring the expansion in directions that seem to lead to the goal, using a heuristic function that indicates how close a node is to the goal. The priority of the queue is based on the sum of both the actual distance from the start and the estimated distance to the goal, computed with the heuristic function. This sum represents an estimate of the total path cost from the start to the goal passing through state s. If the heuristic does not overestimate the cost to the goal, this algorithm is optimal.

D* algorithm [27] has the capability to efficiently replan a new path to the goal considering changes in the environment. It is considered as a generalization of A* for dynamic environments (dynamic A*). It computes optimal paths as new information is delivered. It is efficient since it modifies its previous costs locally (not from scratch). The improved version D* lite [17], is algorithmically simpler and determines the same path than D*. However, both D* and D* lite are not parallelizable.

Dynamic search [28] is based on the wavefront approach, similar to breadth first and *Dijkstra*'s algorithm, expanding in all directions. However, it presents an efficient strategy for repairing trajectories when the environment changes, by reusing previous computations, avoiding so to restart the process of propagation of the wavefront. Moreover, it runs on GPU, leveraging the fact that it is highly parallelizable. The algorithm is also robust to agent movement, to multiple agents and maintains optimality guarantees.

2.3. Selection of the most appropriate path planning algorithm

The high computational power of GPUs has been used in the last decades for many applications that require real time performance. These applications are not limited to graphics processing, but include scientific computation in many fields of knowledge. These applications in video games [13], simulations [29], mobile robotics [30], deep learning [31], among others, evidence a substantial acceleration of the algorithms, especially the ones that do similar work across data points without requiring synchronization, it means, many simple and independent tasks. On the other side, an algorithm with many complex conditions, for example, with nested if-statements, does not leverage the whole capability of a GPU.

Plath planning implemented on GPU have been used for tasks such as inspecting large structures in 3D space [32]and for tackling efficiently multiple agents in large environments [33]. Moreover, the use of path planning algorithms on assistive tools in navigation for blind people is limited to few systems like [2]-[6], so we consider important the advances of path planning algorithms implemented on GPU, for these kinds of applications that require online processing on embedded computers.

Considering the context of the tackled problem, the algorithm selected is dynamic search on GPU [28]. It works in non-uniform and dynamic environments, delivers an optimal path, is robust to changes in the environment and to changes in the position of the user, reuses the previous computations, the solution is complete, admissible and feasible, and is highly parallelizable. The high performance boosted by parallel processing is essential for our purpose of reaching online processing of dense data. Note that D* has similar characteristics than dynamic search but the former is not parallelizable.

2.4. Implementation on GPU of the algorithm for path planning

We implemented and evaluated an algorithm based on dynamic search on GPU [28]. We began with a sequential implementation using MATLAB (for fast prototyping) and C++. Then, we implemented the algorithm on GPU, using CUDA. Finally, we compared their performance and the speed up obtained on GPU, running on a laptop and on the NVIDIA Jetson TX2. Figure 4 shows the pseudocode for computing an optimal path from start node s_s to goal node s_g. This process is repeated for each new start node s_s after the

user moves following the computed path and finishes when the goal is reached by the user or when the algorithm finds that there is not an admissible path to the goal.

Algorithm: Pseudocode for computing an optimal path based on Dynamic Search on GPU

```
Data: Start node s_s, goal node s_g and maps of the environment
Result: Optimal path from start node s_s to goal node s_g
while current node s_c ≠ goal node s_g do
    Get a map;
    if (first map) then
        Init cost;
    else
        Set inconsistencies according to changes of the environment;
    end
    flag_stop = 0;
    while flag_stop == 0 do
        Propagate inconsistency;
        Update the map of cost;
        Compute flag_stop;
    end
end
Draw the optimal trajectory;
```

Figure 4. Pseudocode for path planning based on dynamic search, called efficient dynamic search EDS

The input parameters are the coordinates of the start node s_s, the coordinates of the goal node s_g and information of the environment as an occupancy grid. The result computed by the algorithm is the optimal path from s_s to s_g. Note that the start node changes according to the movement of the user when it follows the trajectory to reach the goal. We manage three maps: map of costs *g*(s), map of predecessors *pred*(s), and map of inconsistencies *incon*(s). Each map has a read only and write only version. Next, we give more details of these maps and of the main processes of the algorithm.

Init cost. For the first occupancy grid of the environment, a map of costs is created. Free cells are assigned a cost of -1, g(s) =-1, representing that these cells need to be updated. Occupied cells are assigned infinite cost, g(s) = inf. The cost of the goal cell is set to zero, $g(s_g)=0$.

Set inconsistencies according to changes of the environment. If a cell changes from free to occupied then the cost in the map is set to infinity and the predecessor to null, since an obstacle has not predecessor. On the other side, if a cell changes from occupied to free then the cost in the map is set to -1, indicating that this cell needs to be updated. In both cases, the map of inconsistencies for this cell is set to 1.

Propagate inconsistency. If the predecessor of a cell is inconsistent then set inconsistency to current cell, delete inconsistency of predecessor and set cost of current cell to -1.

Update the map of costs. We define the parent (predecessor) of any state s' (successor) as the state s that was expanded and that generated s'. On GPU each thread is in charge of updating an individual cell s, by evaluating the neighbors s'.

When a cell is updated, we must use the costs of neighbor cells that have not been updated. Since each thread on GPU reads, modifies and writes a cell of the grid, but the order in which it occurs is undefined, we do not know if we are reading an updated or not updated value. This concurrency problem (race condition) is common in massive parallel processing [34]. For tackling this problem, we use two maps, one for read-only g_r(s) and other for write-only g_w(s). The same solution to race condition is applied to the map of predecessors *pred*(s), and map of inconsistencies *incon*(s).

When the kernel *kernel_planner* is called, the map of cost is updated. The map for read-only g_r(s) will not change, since data will be modified in the map for write-only g_w(s). Once the whole map for write-only has been updated, it is copied to the map for read-only g_r(s). In this way, updated data will be read in the next call of *kernel_planner*.

The *kernel_planner* updates costs of cells for expanding the wavefront and of cells that have been marked as inconsistent. This task makes the algorithm efficient since it avoids to reset the entire map. The expansion of the wavefront goes from the goal to the start cell. In each iteration the cost is updated by minimizing the cost function presented in (1).

$$g(s) = min_{s' \in succ(s) \wedge g(s') \geq 0}(c(s, s') + g(s')) \quad (1)$$

where s' is a successor of s, c(s,s') is a function that computes the cost to move from s' to s, g(s') is the cost to move from goal to s', and g(s) is the cost to move from goal to s. This minimization is carried out with successors s' that are not obstacles ($g(s') \neq inf$), that have been updated and that are not inconsistent ($g(s') \geq 0$).

Draw the optimal trajectory. Once the goal is reached, the trajectory can be obtained by following the least-cost path from the goal to the current cell. This information is recovered from the map of predecessors that points to the parent of each cell. We begin with the predecessor of the start cell and we finish when the predecessor of a cell is the goal cell. Compute flag_stop. The inner while loop finishes when the number of iterations reaches a maximum or when the map of costs does not change any more and the number of inconsistencies does not decrease any more.

3. RESULTS AND DISCUSSION

We evaluated the performance w.r.t. time of two algorithms: 1). Dynamic search restarted when the environment changes (dynamic search from scratch DSS), and 2). Dynamic search that reuses the previous computations when the environment changes (efficient dynamic search EDS, defined in the pseudocode of Figure 4). The sequential implementations of these algorithms were done in MATLAB 2020 and C++, while the parallel implementation was done using CUDA. These algorithms were run on a laptop ASUS that has a processor Intel Core i7-6700HQ of 2,60Ghz x 8, 8GB of RAM memory, a GPU NVIDIA GeForce GTX 960M, CUDA 10.0 and Ubuntu 18.04 as operating system. For the implementation with CUDA, the algorithm was executed in both the laptop ASUS and in the NVIDIA Jetson TX2 developer kit. The embedded computer has the dual-core NVIDIA Denver2 + quad-core ARM Cortex-A57, and the GPU NVIDIA Pascal with 256 CUDA cores and 8GB of memory. This device also has CUDA 10.0 and Ubuntu 18.04.

The map is a grid of 100×100 cells with known obstacles, taken from previous maps got in a real environment (see the paved courtyard of Figure 1(b) and the occupancy grid of Figure 2(b)), and unknown obstacles, got randomly by filling free areas up to a certain percentage (synthetic obstacles). The unknown obstacles are detected by the system if these have been at any moment inside the range of measurement of the system, otherwise these cells are considered free.

Since the user is moving in order to reach the goal cell, a new start cell is defined and an optimal path is computed for each step. From the new start cell the environment is measured so the optimal path considers the lastest changes in the environment (see Figure 3(b)). As the user moves following the optimal path, new obstacles appear and cells switch from free to occupied. These changes are tackled efficiently by dynamic search EDS as we will see next.

The evaluation was carried out for three sizes of the grid and three percentages of unknown obstacles. For computing grids of different sizes, the base grid of 100x100 cells was upsampled twice. As a result, we got two grids of 200x200 cells and 400x400 cells. The percentage of unknown obstacles was set to three values: 20%, 40%, and 60% as shown in Figures 5(a)-(d). The range of measurement was set to 3m for all the experiments.

In Table 1, we present the main results obtained with DSS, such as the number of steps done by the user until the goal cell is reached, the average number of iterations for each step and the average time for each step. Note that the number of steps increases as the percentage of unknown obstacles increases, for the same size of the grid, since the system has to re-plan when the environment changes and a better trajectory is possible. This increase is more significant for a bigger grid since the environment is wider and the range of measurement remains in 3m, so more frequent changes in trajectory must be done.

Figures 6 shows the behavior of the execution time for the implementation in MATLAB, for a grid of 400×400 cells and 40% of unknown obstacles. This behavior is similar to the other implementations of DSS. Note that in Matlab, the time remains around 50 seconds for all the steps. This is due to the fact that the map of costs is restarted for every new step that presents changes in the environment, so the same computations of cost must be done for the whole grid, no matter the closeness of the user to the goal cell.

Table 1. Performance with respect to time of dynamic search from scratch DSS

Grid Size	% of Unknown obstacles	Steps/Iter_avg	Tmatlab_avg Laptop ASUS	Tcpp_avg Laptop ASUS	Tcuda_avg Laptop ASUS	Tcuda_avg Jetson TX2
100×100	20	74/44,2432	520,2ms	14,7980ms	2,3513ms	6,9812ms
	40	78/51,8077	562,5ms	14,9760ms	2,6672ms	8,2426ms
	60	105/49,7810	443,3ms	12,4670ms	2,5662ms	7,7540ms
200×200	20	148/109,5068	5087,0ms	145,71ms	7,1042ms	20,347ms
	40	157/126,3567	5504,8ms	154,79ms	8,0326ms	23,887ms
	60	225/157,8978	6390,8ms	163,39ms	9,9111ms	29,421ms
400×400	20	296/195,2973	32071,5ms	1106,4ms	19,914ms	71,019ms
	40	319/267,6583	42767,9ms	1417,6ms	26,940ms	96,077ms
	60	523/325,7342	46742,9ms	1556,0ms	32,134ms	115,69ms

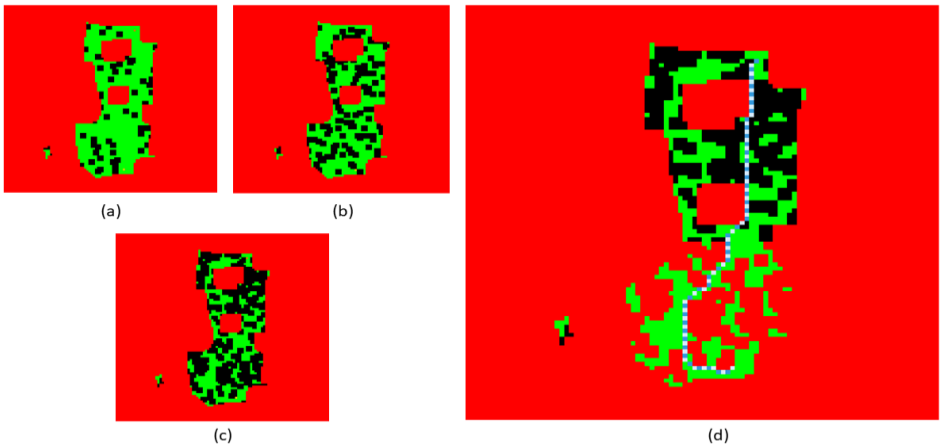


Figure 5. Occupancy grid of 100×100 cells, with occupied cells (red), free cells (green) and unknown obstacles (black), (a) 20%, (b) 40%, (c) 60% of unknown obstacles, and (d) unknown obstacles that are inside the region of measurement of the system, change to occupied cells. The other unknown obstacles are considered free cells from that pose of the user. The start cell is at the bottom while the goal cell is on the top of the grid. The trajectory for that step was drawn in blue and white

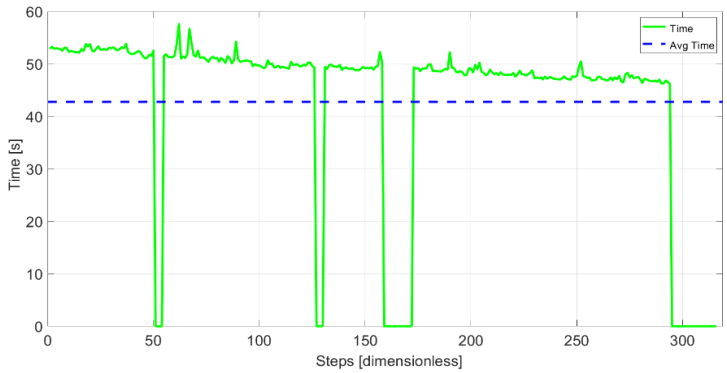


Figure 6. Time for each step, for the algorithm DSS running on the laptop and implemented with MATLAB, the grid has 400×400 cells and 40% of unknown obstacles

The average time of execution for a step of the algorithm in MATLAB is much greater compared with the average time in C++ and CUDA as shown in Figure 7. This is so because MATLAB is not a compiled language but an interpreted one. The advantage of MATLAB is the easiness to prototype code, so it was used for that task. Comparing the time of the implementations in C++ and CUDA, we can see an improvement with the last one, that is significant in real-time applications like in this case.

Finally, we compared the implementation with CUDA, running on both the laptop ASUS and on the NVIDIA Jetson TX2. Note that the last one is slower than the former since it has less CUDA cores (256 compared to 640 for the GeForce GTX 960M). Although the algorithm runs slower in the embedded platform and that the algorithm does not reuses previous computations of cost, the time reached is good for working with grid sizes up to 200×200 cells (less than 30ms). In this case, the planner must be executed with an appropriate interval of time, for example, one second between two consecutive steps. This time between steps would allow the user to move to a new position, following the computed path, before computing the next one. In Table 2, we present the main results obtained with EDS, such as the number of steps done by the user until the goal cell is reached, the average number of iterations for each step and the average time for each step. The range of measurement was kept constant (3m) for all the grids. Note that the number of steps is equal to DSS. This means that both algorithms found the same trajectory in each step until reaching the goal. Moreover, the average number of iterations for each step decreased and it is reflected in the reduction of the time for all implementations of EDS, compared to the respective implementation of DSS. It is because EDS reuses previous computations of cost, unlike DSS that restarts the map of costs for each new step. This reduction in time is more evident for steps with a start cell far from the goal (first steps) as can be seen in Figure 8, since the inconsistencies are generated closer to the border and the propagation finishes faster.

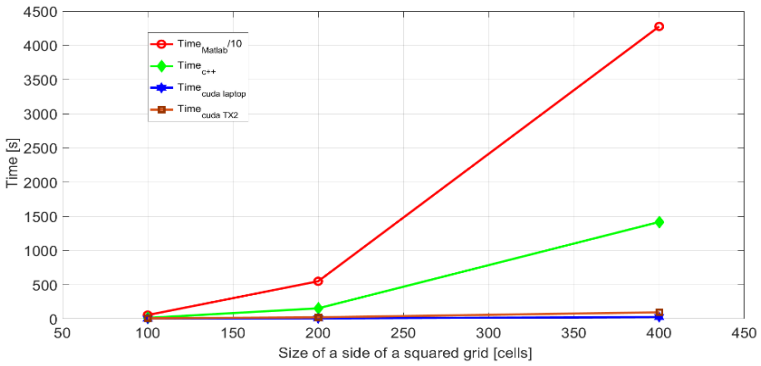


Figure 7. Comparison of execution time for the algorithm DSS implemented in MATLAB (red), C++ (green) and CUDA (blue and orange). The execution time for the implementation in MATLAB was scaled for improving the visual comparison. The implementation with CUDA was executed in both the laptop and the NVIDIA Jetson TX2 (orange). The grid has 40% of unknown obstacles.

Table 2. Performance with respect to time of EDS

Grid Size	% of Unknown obstacles	Steps/Iter_avg	Tmatlab_avg Laptop ASUS	Tcpp_avg Laptop ASUS	Tcuda_avg Laptop ASUS	Tcuda_avg Jetson TX2
100×100	20	74/11,6351	119,4ms	5,1022ms	2,0622ms	5,0335ms
	40	78/16,4615	164,4ms	7,8480ms	3,031ms	7,7004ms
	60	105/16,0000	147,0ms	7,0913ms	2,9808ms	7,2748ms
200×200	20	148/19,5608	958,9ms	39,578ms	4,1793ms	11,819ms
	40	157/30,1083	1416,8ms	57,691ms	6,4539ms	17,588ms
	60	225/27,5378	1143,2ms	48,395ms	5,8261ms	16,040ms
400×400	20	296/35,7568	7121,1ms	291,8ms	13,814ms	42,637ms
	40	319/58,2382	9758,8ms	450,2ms	22,215ms	67,354ms
	60	523/59,9828	9056,8ms	423,8ms	22,312ms	69,103ms

The implementation of EDS in MATLAB takes much more time to execute than the implementations in C++ and CUDA as shown in Figure 9(a), as happened with DSS. The execution time for the implementation in C++ is good for small grids. For a grid of 400x400 cells, the time increases too much and could hinder a real time application. Conversely to the implementations in MATLAB and C++, the implementation in the laptop using CUDA, takes a small time to execute, even for a big grid like the one with 400x400 cells (less than 25ms). The implementation in the NVIDIA Jetson TX2 with CUDA, is outstanding compared to the implementations in MATLAB and C++. Although the time is bigger than the implementation in the laptop with CUDA, it is less than 18ms, for grid sizes up to 200x200 cells.

Figure 9(b) shows a comparison in performance w.r.t. time for implementations with CUDA, running on the NVIDIA Jetson TX2, of DSS and EDS. The improvement in time obtained with EDS allows the system to reduce the interval of time between two steps, so the path to the goal can be updated faster, according to changes in the environment.

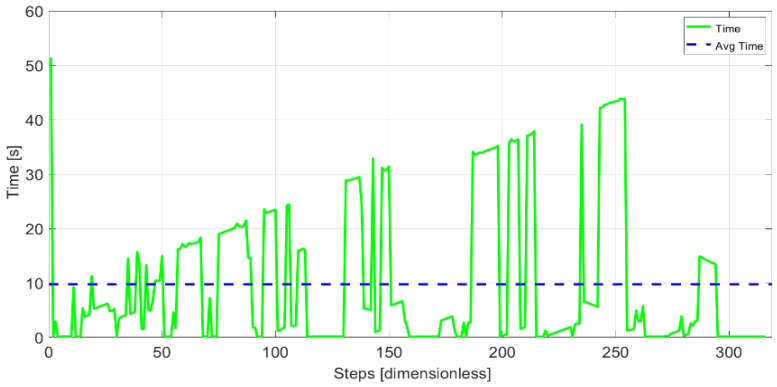


Figure 8. Time for each step, for the algorithm EDS running on the laptop and implemented with MATLAB. The grid has 400×400 cells and 40% of unknown obstacles

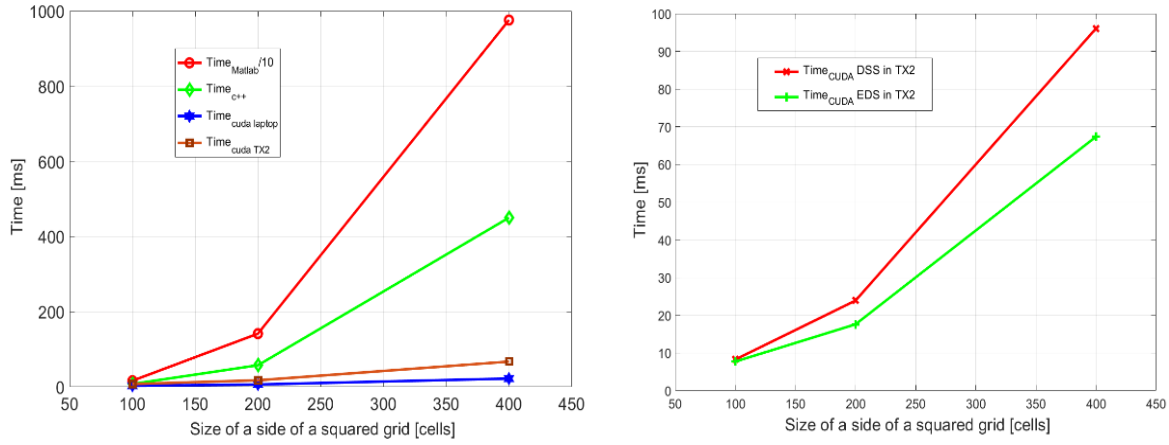


Figure 9. Comparison of performance w.r.t. time of the algorithms (a) comparison of execution time for the algorithm EDS implemented in MATLAB (red), C++ (green) and CUDA (blue and yellow). The execution time for the implementation in MATLAB was scaled for improving the visual comparison. The implementation with CUDA was executed in both the laptop and the NVIDIA Jetson TX2 (orange). The grid has 40% of unknown obstacles and (b) comparison of execution time for the algorithms DSS (red) and EDS (green) implemented with CUDA and executed in the NVIDIA Jetson TX2

Finally, we compare the time for path planning of [2] with our planner, considering a grid of size 200×200 cells. The former reports a processing time in a laptop between 15ms and 30ms, for most of the frames, while ours takes 6,45ms and 17,59ms on average, in a laptop and in the Jetson TX2, respectively. These times indicate that our algorithm, based on [28], outperforms the planner of [2] in similar conditions.

4. CONCLUSION

In this work, we presented the requirements for implementing a path planning algorithm applied to a real problem of assisting blind people to reach a goal in an optimal and safe way. The requirements found are: dynamic and partially known environment, non-uniform cost, moving user, static goal, real time processing of data and immediate feedback. We implemented and evaluated the performance w.r.t time of a sequential and a parallel implementation of a path planner based on the algorithm called dynamic search. The sequential implementation was done in MATLAB and C++ and executed in a laptop, while the parallel one was done with CUDA and executed in both a laptop with GPU and in the NVIDIA Jetson TX2. The algorithm has two versions. The first one is dynamic search from Scratch DSS that restarts the computations of the cost for the whole grid for each step while the second one is Efficient dynamic search EDS that reuses previous computations of cost.

The execution times are highly dependent on the complexity of the environment such as the size of the grid and the percentage of unknown obstacles, and the range of measurement. We evaluated the algorithms for three different sizes of the grid and for three percentages of unknown obstacles, keeping the range of measurement constant, setting to a value similar to the range of the real system. Our parallel implementation of DSS, for the grid of 400x400 cells and 40% of unknow obstalces, improves the performance up to 52,62 times in the laptop and 14,75 times in the TX2, compared with the implementation on CPU (in C++). Our parallel implementation of EDS improves the performance up to 20,26 times in the laptop and 6,68 times in the TX2, compared with the implementation on CPU (in C++). Comparing the implementations of DSS and EDS with CUDA, the last one has a speed up of 1,21 when is executed in the laptop and 1,42 when is executed in the TX2. Finally, comparing EDS executed in the laptop with EDS executed in the TX2, we found that the embedded platform is slower with a slowdown of 3,10 times. However, both implementations in CUDA of EDS and DSS running on the TX2, have good performance, with execution time less than 18ms for EDS, and less than 30ms for DSS, for sizes of the grid up to 200x200 cells. This performance of the module of path planning allows us to integrate it, in a future work, to our main system for purposeful navigation of blind people, obtaining real time replanning in dynamic environments.

ACKNOWLEDGEMENTS

We would like to thank Google (Latin America Research Awards LARA 2018) that gave us financial support during one year. We also would like to thank Fundación CEIBA, a national Foundation that supports this research since September 2019, together with Universidad Nacional Abierta y a Distancia UNAD.

REFERENCES

[1] E. Brady, M. R. Morris, Y. Zhong, S. White, and J. P. Bigham, "Visual challenges in the everyday lives of blind people," in *Conference on Human Factors in Computing Systems - Proceedings*, Apr. 2013, pp. 2117–2126, doi: 10.1145/2470654.2481291.

[2] Y. H. Lee and G. Medioni, "Wearable RGBD indoor navigation system for the blind," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8927, Springer International Publishing, 2015, pp. 493–508.

[3] M. I. Islam, M. M. Raj, S. Nath, M. F. Rahman, S. Hossen, and M. H. Imam, "An indoor navigation system for visually impaired people using a path finding algorithm and a wearable cap," *Proc. 3rd Int. Conf. for Convergence in Tech. (I2CT)*, Apr. 2018, doi: 10.1109/I2CT.2018.8529757.

[4] Q. Chen *et al.*, "CCNY Smart Cane," in *2017 IEEE 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems, CYBER 2017*, Jul. 2018, pp. 1246–1251, doi: 10.1109/CYBER.2017.8446303.

[5] Y. Zhang, Y. Zhao, T. Wei, and J. Chen, "Dynamic path planning algorithm for wearable visual navigation system based on the improved A," in *IST 2017 - IEEE International Conference on Imaging Systems and Techniques, Proceedings*, Oct. 2017, vol. 2018-January, pp. 1–6, doi: 10.1109/IST.2017.8261468.

[6] H. Zhang and C. Ye, "Human-robot interaction for assisted wayfinding of a robotic navigation aid for the blind," in *International Conference on Human System Interaction, HSI*, Jun. 2019, vol. 2019-June, pp. 137–142, doi: 10.1109/HSI47298.2019.8942612.

[7] C. Campos, R. Elvira, J. J. G. Rodriguez, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM3: an accurate open-source library for visual, visual-inertial, and multimap SLAM," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, Dec. 2021, doi: 10.1109/TRO.2021.3075644.

[8] J. Klečka, K. Horák, and O. Boštík, "General concepts of multi-sensor data-fusion based SLAM," *IAES International Journal of Robotics and Automation (IJRA)*, vol. 9, no. 2, p. 63, Jun. 2020, doi: 10.11591/ijra.v9i2.pp63-72.

[9] Y. Li and S. B. Lang, "A stereo-based visual-inertial odometry for SLAM," in *Proceedings - 2019 Chinese Automation Congress, CAC 2019*, Nov. 2019, pp. 594–598, doi: 10.1109/CAC48633.2019.8997432.

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: unified, real-time object detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2016, vol. 2016-December, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[11] N. Rachburee and W. Punlumjeak, "An assistive model of obstacle detection based on deep learning: YOLOv3 for visually impaired people," *International Journal of Electrical and Computer Engineering*, vol. 11, no. 4, pp. 3434–3442, Aug. 2021, doi: 10.11591/ijece.v11i4.pp3434-3442.

[12] M. Niazi-Razavi, A. Savadi, and H. Noori, "Toward real-time object detection on heterogeneous embedded systems," in *2019 9th International Conference on Computer and Knowledge Engineering, ICCKE 2019*, Oct. 2019, pp. 450–454, doi: 10.1109/ICCKE48569.2019.8964764.

[13] A. J. O. Vargas, J. E. C. Serrano, L. C. Acuna, and J. C. Martinez-Santos, "Path planning for non-playable characters in arcade video games using the wavefront algorithm," in *Proc. IEEE Games, Multimedia, Animation and Multiple Realities Conf. (GMAX)*, 2020, doi: 10.1109/GMAX49668.2020.9256835.

[14] P. Shruthi and R. Resmi, "Path planning for autonomous car," in *2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies, ICICICT 2019*, Jul. 2019, pp. 1387–1390, doi: 10.1109/ICICICT46008.2019.8993295.

[15] X. Wang, X. Yu, and W. Sun, "A path planning and tracking control for autonomous vehicle with obstacle avoidance," in *Proceedings-2020 Chinese Automation Congress, CAC 2020*, Nov. 2020, pp. 2973–2978, doi: 10.1109/CAC51589.2020.9327065.

[16] A. S. Huang *et al.*, "Visual odometry and mapping for autonomous flight using an RGB-D camera," in *Springer Tracts in Advanced Robotics*, vol. 100, Springer International Publishing, 2017, pp. 235–252.

[17] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, Jun. 2005, doi: 10.1109/TRO.2004.838026.

[18] R. L. Campos, R. Jafri, S. A. Ali, O. Correa, and H. R. Arabnia, "Evaluation of the google tango tablet development kit: A case study of a localization and navigation system," in *Proceedings - 2018 International Conference on Computational Science and Computational Intelligence, CSCI 2018*, Dec. 2018, pp. 501–506, doi: 10.1109/CSCI46756.2018.00103.

[19] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968, doi: 10.1109/TSSC.1968.300136.

[20] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, Oct. 1973, doi: 10.3138/fm57-6770-u75u-7727.

[21] E. A. Hansen and R. Zhou, "Anytime heuristic search," *Journal of Artificial Intelligence Research*, vol. 28, pp. 267–297, Mar. 2007, doi: 10.1613/jair.2096.

[22] H. Zhang and C. Ye, "An indoor navigation aid for the visually impaired," in *2016 IEEE International Conference on Robotics and Biomimetics, ROBIO 2016*, Dec. 2016, pp. 467–472, doi: 10.1109/ROBIO.2016.7866366.

[23] A. A. Diáz-Toro, S. E. Campanã-Bastidas, and E. F. Caicedo-Bravo, "Vision-based system for assisting blind people to wander unknown environments in a safe way," *Journal of Sensors*, vol. 2021, pp. 1–18, Mar. 2021, doi: 10.1155/2021/6685686.

[24] H. Hakim, Z. Alhakeem, and S. Al-Darraji, "Goal location prediction based on deep learning using rgb-d camera," *Bulletin of Electrical Engineering and Informatics*, vol. 10, no. 5, pp. 2811–2820, Oct. 2021, doi: 10.11591/eei.v10i5.3170.

[25] E. F. Moore, *The shortest path through a maze*. New York: Bell Telephone System., 1957.

[26] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959, doi: 10.1007/BF01386390.

[27] A. Stentz, "The focussed D* algorithm for real-time replanning," in *Proc. 14th Int. Joint Conf. on Artif. Intelligence IJCAI*, 1995, pp. 1652–1659.
[28] M. Kapadia, F. Garcia, C. D. Boatright, and N. I. Badler, "Dynamic search on the GPU," in *IEEE International Conference on Intelligent Robots and Systems*, Nov. 2013, pp. 3332–3337, doi: 10.1109/IROS.2013.6696830.
[29] S. Lim and P. Kang, "Implementing scientific simulations on GPU-accelerated edge devices," in *International Conference on Information Networking*, Jan. 2020, vol. 2020-January, pp. 756–760, doi: 10.1109/ICOIN48656.2020.9016467.
[30] B. Yan, J. Xin, M. Shan, and Y. Wang, "CUDA implementation of a parallel particle filter for mobile robot pose estimation," in *Proceedings of the 14th IEEE Conference on Industrial Electronics and Applications, ICIEA 2019*, Jun. 2019, pp. 578–582, doi: 10.1109/ICIEA.2019.8833856.
[31] E. Buber and B. Diri, "Performance analysis and CPU vs GPU comparison for deep learning," ," in *Proc. 6th Int. Conf. on Control Eng. & Info. Tech. (CEIT)*, Oct. 2018, doi: 10.1109/CEIT.2018.8751930.
[32] R. Almadhoun, T. Taha, L. Seneviratne, J. Dias, and G. Cai, "GPU accelerated coverage path planning optimized for accuracy in robotic inspection applications," in *Midwest Symposium on Circuits and Systems*, Oct. 2016, vol. 0, doi: 10.1109/MWSCAS.2016.7869968.
[33] S. Mufti, V. Roberge, and M. Tarbouchi, "A GPU accelerated path planner for multiple unmanned aerial vehicles," in *Proc. IEEE Canadian Conf. of Electrical and Comp. Eng. (CCECE)*, May 2019, doi: 10.1109/CCECE.2019.8861921.
[34] M. Ahmad and O. Khan, "GPU concurrency choices in graph analytics," in *Proceedings of the 2016 IEEE International Symposium on Workload Characterization, IISWC 2016*, Sep. 2016, pp. 178–187, doi: 10.1109/IISWC.2016.7581278.

BIOGRAPHIES OF AUTHORS

Andrés Díaz-Toro received his B. Sc degree in Electronic Engineering from Universidad de Nariño, Pasto, Colombia, in 2009, the M.Sc. degree in Automation and the Ph.D. degree in Electronic Engineering from Universidad del Valle, Cali, Colombia, in 2012 and 2018, respectively. He has been assistant professor of undergraduate and graduate courses in Universidad del Valle. In 2018 he was awarded with Google Reseach Awards for Latin America with a project related to assistive tools for blind people. Currently, he is working as professor and researcher in Universidad Nacional Abierta y a Distancia UNAD, with a postdoctoral project for assisting blind people in purposeful navigation. His research interests include mobile robotics, computer vision, artificial intelligence, process control, optimization techniques and applications at real time with embedded computers. He can be contacted at email: andres.diaz@unad.edu.co.

Paula Mosquera-Ortega is a last-year Master student of management of information technology in Universidad Nacional Abierta y a Distancia UNAD. She completed her B.Sc. degree in Mathematics, from Universidad de Nariño, in 2001. Moreover, she obtained the degree as Specialist in Pedagogy of Creativity in the same University, in 2014. Currently, she is working in the project "Methods for evaluating the performance of path planning algorithms focused to assist blind people". Since 2014, she has been working as assistant professor in mathematical science in several universities in Colombia, such as Universidad de Nariño, CESMAG, ESAP and UNAD. Currently, she is working as assistant professor in mathematical science in UNAD and as a researcher in the Davinci group. She can be contacted at email: paula.mosquera@unad.edu.co.

Gustavo Herrera-Silva is a last-year student of B.Sc. in Electronic Engineering in Universidad Nacional Abierta y a Distancia UNAD. He is part of the Davinci research group and of the seedbed for reaserch SeInTec Tesla at UNAD. His research interests include mobile robotics, artificial intelligence, industrial automation, and teleinformatics, information systems development, and telecommunications. He is currently working on the project "Implementation on GPU of a Path Planning Algorithm to Assist Blind People in Purposeful Navigation at Real Time and in Dynamic Environments". He can be contacted at email: gaherrerasi@unadvirtual.edu.co.

Sixto Campaña-Bastidas has a B.Sc degree in Systems Engineering from Universidad Mariana, Pasto, Colombia, a M.Sc. degree in Telematic networks and services, from Universidad del Cauca, Popayan, Colombia, a M.Sc. degree in free software from Universidad Autónoma de Bucaramanda, Colombia, and a Ph.D. degree in Telecommunications from Universidad Pontificia Bolivariana, Medellín, Colombia. He has several postdoctoral stays in Spain and Sweden and is an International Intern for the European Union. Currently, he is working as associated professor and researcher in Universidad Nacional Abierta y a Distancia UNAD. In 2018 he was awarded with Google Reseach Awards for Latin America with a project related to Mothernet Project. His research interests include Internet of things, Machine Learning, process control, optimization techniques and applications at real time with embedded computers. He can be contacted at email: sixto.campana@unad.edu.co.