Autonomous mapping and navigation for a holonomic robot in an unknown environment

Alejandro Dumas Leon[a], Jorge Tomás Araujo González[a], Eduardo Arturo Mendoza Gómez[a], Ulises Orozco-Rosas[a,*], and Kenia Picos[a]

[a]CETYS Universidad, Ave. CETYS Universidad No. 4, Fracc. El Lago, C.P. 22210, Tijuana, Baja California, México

ABSTRACT

This work presents the development of an autonomous mapping and navigation system tailored for a holonomic robot operating in unknown environments, leveraging Light Detection and Ranging (LiDAR) technology. The core of this research lies in integrating a sophisticated Simultaneous Localization and Mapping (SLAM) algorithm with real-time LiDAR data processing and the GPU's parallel capabilities to process several subspaces concurrently. This work introduces a novel method to accelerate route planning in an omnidirectional mobile robot fusing advances in hardware with sophisticated algorithmic techniques, a new paradigm is established in path planning for omnidirectional mobile robots, marking an important milestone in the search for more agile and capable robotic systems.

Keywords: Iterative deepening approach, mobile robots, parallel computing, path planning, state space.

1. INTRODUCTION

Pursuing faster and more efficient algorithms is a constant quest in the ever-evolving landscape of computational science and problem-solving. Among the many challenges researchers and engineers face, path planning within a complex state space is a formidable puzzle. This intricate problem, with its numerous dimensions and complicated calculations, has long demanded innovative approaches to achieve practical solutions. One such approach, at the forefront of contemporary computational strategies, seeks to harness the immense computational power of the Graphics Processing Unit (GPU). This groundbreaking technique addresses the computational complexity of path planning and offers the tantalizing prospect of unlocking unprecedented speeds and efficiencies. The core principle revolves around dividing the intricate state space into more manageable subspaces. An interactive depth search is applied with meticulous precision within these subspaces.

What sets this approach apart is the remarkable parallel processing prowess inherent to GPUs. Unlike traditional Central Processing Units (CPUs), GPUs consist of numerous cores that considerably increase the number of tasks that can be performed concurrently. This inherent parallelism allows concurrently handling multiple subspaces, fundamentally transforming the path-planning process. Furthermore, using shared memory within the GPU is a crucial catalyst in this computational alchemy. By storing essential data within this shared memory, access times to the global memory are dramatically reduced, optimizing the computational workflow. The cumulative effect of these innovations results in a seismic shift in path planning speed, offering a quantum leap beyond what traditional CPU-based executions can achieve.[1]

The study's novelty resides in its comprehensive approach, combining precise LiDAR mapping, omnidirectional mobility, and autonomous pathfinding to navigate in static environments. The initial phase involves the design of the robot's holonomic movement system and the integration of the LiDAR sensor. Subsequent stages focus on developing a SLAM algorithm tailored for LiDAR data, enabling the robot to create accurate maps of its surroundings and determine its position within these maps. The last component involves implementing algorithms for real-time path planning and obstacle avoidance. The research methodology includes rigorous testing

*Further author information:
U. Orozco-Rosas: E-mail: ulises.orozco@cetys.mx

in varied environments to validate the system's effectiveness, with results indicating significant advancements in autonomous robotics navigation.[2,3]

Nevertheless, as with any transformative technology, the full realization of GPU-accelerated path planning demands meticulous optimization and parameter tuning. The harnessing of the GPU's vast computational potential is a nuanced art that requires finesse and expertise. Researchers and practitioners alike must navigate the intricate terrain of algorithmic optimization and hardware utilization to unlock the GPU's unparalleled capacity fully. In this realm of computational innovation, the fusion of state space, iterative deepening approach, and GPU acceleration represent a powerful trifecta that has the potential to redefine the boundaries of what is computationally achievable. It is a testament to the relentless pursuit of efficiency and the ceaseless exploration of novel approaches to complex problems. As we delve deeper into the intricacies of this groundbreaking technique, we embark on a journey into the heart of computational discovery, where the boundaries of what is possible continue to expand, setting new standards in path planning and beyond.

The task is to perform pathfinding in a state space, initially sequentially and ultimately in parallel, using the GPU. Therefore, path planning in state space using the iterative deepening approach on a GPU (specifically the CUDA programming language) can provide a faster and more efficient solution for this complex problem. This could be highly beneficial in robotics and autonomous systems applications such as route planning.[4,5] Using the iterative depth method, the main objective of this work is to employ parallel computing with the GPU in an omnidirectional mobile robot to perform path planning in state space.

The remainder of this paper is organized as follows: Section 2 briefly describes the theoretical framework. Section 3 explains the proposed implementation. Section 4 presents the experimental procedure and then the results obtained. Finally, Section 5 presents the general conclusions.

2. THEORETICAL FRAMEWORK

This section delves into the theoretical foundations that underpin the study, concentrating on a spectrum of concepts crucial for a comprehensive understanding of the research approach. Central to this exploration is "Path Planning," a key element in disciplines like robotics and computer science, which involves identifying an optimal route or strategy.[6] The "Iterative Deepening Approach" is also examined, an algorithmic technique that skillfully balances depth and breadth in search processes, particularly pertinent in environments with vast state spaces. The notion of "State Spaces," embodying all conceivable configurations or conditions in a given problem, is fundamental to this discourse, providing a structure for algorithmic problem-solving. Additionally, the increasing importance of "Parallel Programming" in enhancing computational efficiency is acknowledged as a vital facet of contemporary computing paradigms. Within this context, "CUDA," NVIDIA's parallel computing platform and programming model, is highlighted for its transformative impact on handling computing tasks, especially in high-performance computing scenarios. This section aims to provide a thorough background, laying the groundwork for the subsequent application and examination of these concepts within the specific research field.

2.1 Trajectory Planning

Trajectory planning is a fundamental concept in robotics and control systems, focusing on determining the optimal path a robot should follow over time. It involves not only the selection of a path from one point to another but also the consideration of how an entity moves along this path, factoring in its speed, acceleration, and other dynamic aspects.[7] The main objective of trajectory planning is to develop a sequence of movements or a trajectory that allows the robot to perform its task efficiently and in compliance with its physical and operational constraints.[8] As Steven LaValle's book "Planning Algorithms" detailed, trajectory planning primarily involves solving two interrelated problems. First, it requires the determination of a path that the robot can follow, avoiding obstacles and ensuring feasibility within its operational environment. This path is often calculated in a state space, denoted as X, where each state x comprises the robot's configuration (q) and its velocity (q'). The second aspect of trajectory planning involves figuring out the appropriate velocities and accelerations (q') at each point along the path, ensuring that these movements respect the robot's mechanical limitations and differential constraints.[9]

2.2 Iterative Deepening Approach

A tree search algorithm that explores different depths to find an optimal or satisfactory solution. Starting from a root node, successor nodes are recursively explored. Iterative depth involves conducting a depth search iteratively, starting with a given maximum depth and gradually increasing this maximum in each iteration until the desired solution is found. This efficient and memory-conservative method makes it suitable for large search spaces. The idea is to use a depth-first search and find all states that are distance i or less from x_1. If the goal is not found, the previous work is discarded, and depth-first is applied to find all states of distance $i + 1$ or less from x_1.

2.3 State Space

It encompasses all conceivable scenarios within a given context. This could range from the position and orientation of a robot, the locations of tiles in a puzzle, to the position and velocity of a helicopter. LaValle emphasizes the versatility of state spaces, acknowledging that they can be either discrete finite, countably infinite, or continuous, which means uncountably infinite. A critical aspect highlighted by LaValle is the implicit representation of the state space in most planning algorithms. Due to the vast number of states or their combinatorial complexity, it is impractical to explicitly represent the entire state space in most applications. Nevertheless, defining the state space remains a crucial step in formulating planning problems and designing and analyzing algorithms to solve them. LaValle advises careful definition of the state space in specific applications, ensuring that irrelevant information is not encoded into a state, thereby maintaining the efficiency and relevance of the planning algorithm. This thoughtful approach to defining state spaces underscores their importance in the development and application of planning algorithms.[9]

2.4 Path Planning

Path planning in robotics is the problem of finding the shortest and most feasible path (route) from point A to point B, this happens because moving less implies saving energy and time. Thanks to computers many algorithms have been created to give a solution to that problem, some examples are Dijkstra's algorithm, A^*, D^*, artificial potential fields, iterative deepening approach, among others. For the present work, the Iterative Deepening approach was chosen since it is an algorithm that can be parallelized due to subtrees that are formed.

The implementation requires many aspects to be considered, for example, the grid to be used, the robot that will follow the resulting path, and the resolution of the path. Iterative Deepening Approach is a search strategy that combines the depth-first search's (DFS) memory efficiency and the breadth-first search's (BFS) completeness. It involves incrementally deepening the search depth limit until the goal is found. The Time Complexity is $O(b^d)$, where b is the branching factor and d is the depth of the goal node. The time complexity can be significantly reduced by parallelizing the search process. If the search tree is divided among p processors, the theoretical time complexity can be reduced to $O(b^d/p)$.

There are many possible configurations for the grid which represents the graph for the algorithm to be implemented, the grid could have been a tile or a hex configuration. However, the octile configuration was chosen because of its performance compared with the simple tile movement to North, South, East, and West (N, S, E, W) and the simplicity of representing the grid in real life compared to the hex configuration. The octile allows movement in 8 directions which are North, South, East, West, Northeast, Northwest, Southeast, and Southwest (N, S, E, W, NW, NE, SE, SW).

2.5 Parallel Programming

This involves designing computer programs and algorithms for concurrent execution on multiple processors or cores, enhancing speed and efficiency. Traditional sequential programming executes instructions one after the other, limiting performance in data-intensive or computationally heavy tasks. Parallel programming leverages multiple processing units concurrently to expedite task execution.[1] One of the fundamental concepts in parallel programming is task decomposition. It involves breaking down a larger computational task into smaller, more manageable subtasks that can be executed concurrently. The key challenge here is to ensure that these sub-tasks can run independently and efficiently without causing conflicts or bottlenecks. Effective task decomposition is essential for harnessing the full potential of parallelism.[10]

Parallel programming encompasses various forms of parallelism, each suited to several types of problems. Task-level parallelism divides a program into smaller units of work that can run in parallel. Data-level parallelism involves distributing data across multiple processing units and performing identical operations on different data elements simultaneously. Instruction-level parallelism exploits parallelism within a single instruction stream, often seen in modern processors with pipelining or superscalar execution capabilities.[1]

Parallel programming models can be categorized into two main types: shared memory and distributed memory. Shared memory parallelism involves multiple threads or processes sharing a common memory space, allowing them to communicate by reading and writing to shared data structures.[11] In contrast, distributed memory parallelism relies on message passing, where processes or nodes communicate through explicit messages, each having its private memory space. Distributed memory models are commonly used in High-Performance Computing (HPC) environments with frameworks like Message Passing Interface (MPI). In parallel programming, synchronization mechanisms are crucial to coordinate the execution of concurrent tasks. These mechanisms ensure that tasks do not interfere with each other or access shared resources simultaneously, which could lead to data corruption or race conditions. Common synchronization primitives include locks, semaphores, barriers, and atomic operations, helping to maintain the order and integrity of parallel execution.

Efficient parallel programs require load balancing to distribute the workload evenly among available processors or cores. Load imbalances can lead to situations where some processors are idle while others are overwhelmed, resulting in suboptimal performance. Achieving good scalability, where the program's performance scales with the number of available processing units, is a significant challenge in parallel programming and often requires careful design and optimization.[12]

2.6 CUDA

NVIDIA's parallel computing platform allows general-purpose computing on graphics processing units (GPUs). Modern GPUs are highly parallel architectures capable of executing multiple computational tasks simultaneously, unlike traditional GPUs designed solely for graphics rendering.

2.7 LiDAR

The LiDAR sensor is a powerful sensing device that has had a great impact on the robotics industry thanks to the possibility of offering quick and precise distance data at a relatively low price and high-speed processing capabilities. In this section, the use of a 360^o RPLiDAR A2M8[13] (shown in Fig. 1) for obstacle detection and path re-planning will be explained regarding the objective of the project and the respective specifications of the model chosen will be shown as well.

Figure 1: RPLiDAR A2M8

Obstacle detection and avoidance are fundamental problems discussed in mobile robotics and autonomous vehicles. Since moving through a safe path is fundamental for navigation from a starting state to a goal state efficiently.[14] Different technologies developed throughout the years are useful for solving the obstacle avoidance problem, for example, cameras are really powerful for obstacle detection and identification thanks to computer vision algorithms but they are quite expensive and require a lot of hardware; there are other sensors for just measuring distances which properly processed can be useful for obstacle detection, for example, the ultrasonic

sensor, this sensor is easy to acquire and is accessible, however, the data to be processed has bad quality and for precision purposes, the effectiveness of these sensors is limited. A good option is the LiDAR sensor, which is more precise than the ultrasonic, cheaper, and faster than a camera, this sensor can be found as a 2D or 3D sensor, the first one is quite useful for measuring distances in a plane while the second one has those capabilities plus the ability to produce very precise point clouds.[15] For this project, a 2D LiDAR was considered, as shown in Fig. 2.

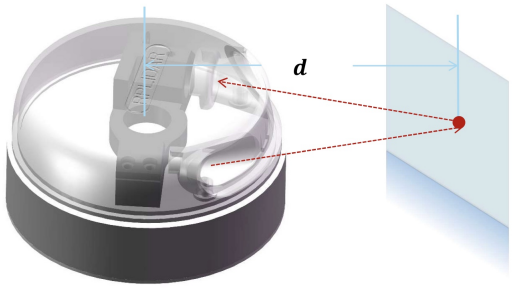


Figure 2: RPLiDAR working schematic. Image by Slamtec.

The algorithm discussed in this paper implies the ability of a differential robot to find a path from a starting coordinate to a goal coordinate and re-plan in case there is a new obstacle blocking the generated path. The LiDAR will be aimed at the front of the robot and measure the distance considering the respective angle (see Fig. 3.), in case the robot senses something positioned in less than a robot's step, the re-planning algorithm will be activated.

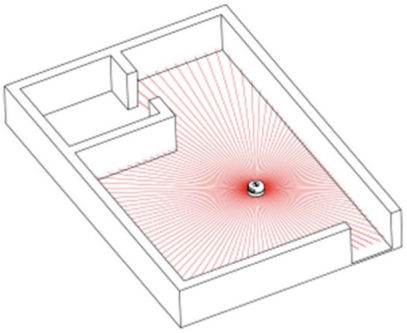


Figure 3: RPLiDAR Scanning. Image by Slamtec.

A list of the specifications of interest regarding the RPLiDAR used, provided by SLAMTEC,[13] is shown below.

- Distance measurement range: 0.20 to 12 meters.
- Angular range: 360 degrees clockwise.
- Distance resolution: Minor than 0.5 mm.
- Angular resolution: 0.45 (min.), 0.9 (typical), 1.35(max.).
- Sample frequency: 2000 Hz (min.), 8000 Hz (typical), 8010 Hz (max.).
- Scan rate: 5 Hz (min.), 10 Hz (typical), 15 Hz (max.).
- The sensor uses 3.3V-TTL serial port (UART) communication for sending the data to any microcontroller.

3. PROPOSAL

Fig. 4 delineates the sequential stages involved in CUDA-based computation, starting from the allocation of memory on the host (CPU) and the device (GPU), through to the transfer of data between host and device.

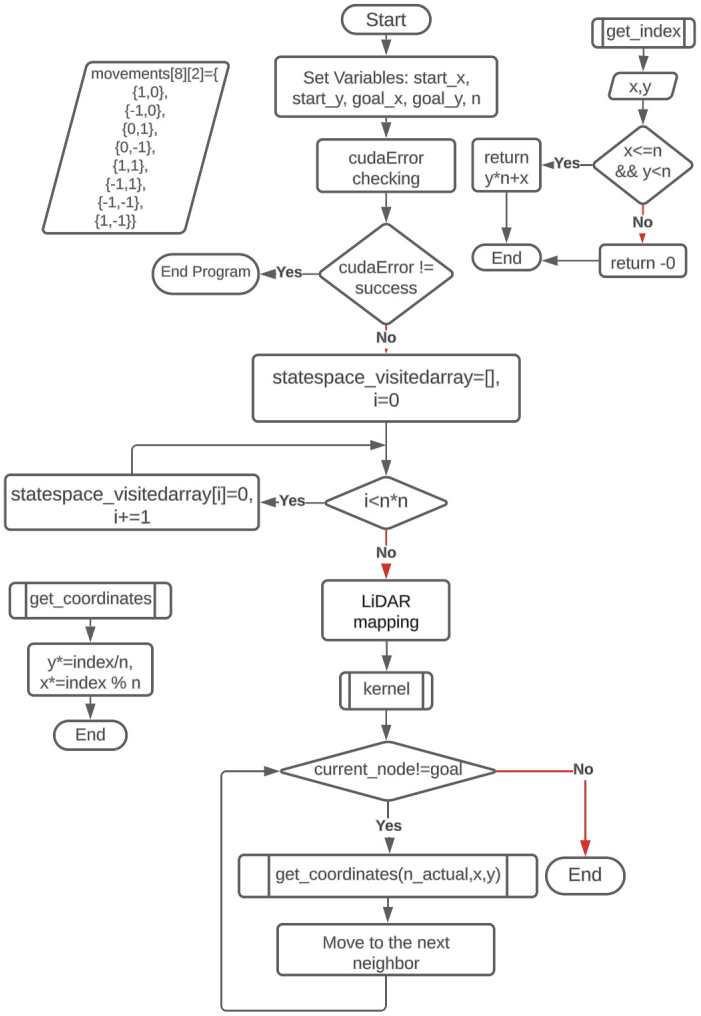


Figure 4: Main program diagram.

Fig. 5 details the execution of parallel kernels on the GPU, displaying the distribution of tasks across multiple threads and blocks within the GPU's architecture. Following the kernel execution, the diagram illustrates the process of transferring the computed results back to the host.

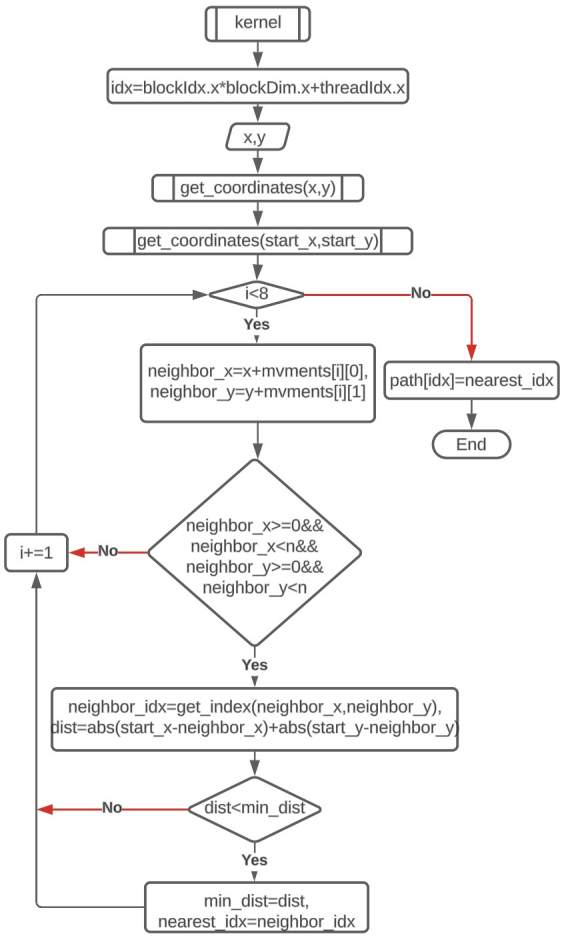


Figure 5: CUDA kernel diagram.

The main functions of the proposed system can be described as follows:

- **Set Initial State and Goal State**. In Fig. 6, the initial state marks the beginning point, defining the exact location and orientation from which the journey or operation will start. This state is pivotal as it serves as the cornerstone for all subsequent path calculations. Following this, the goal state is established, delineating the intended destination or endpoint. This goal is crucial as it guides the trajectory and final aim of the path-planning algorithm. By accurately setting these elements, the system is equipped to utilize parallel computing techniques, enabling it to concurrently explore multiple pathways and scenarios, significantly enhancing the efficiency and effectiveness of the path-planning solution.
- **Set hardware peripherals.** Configuring and initializing external devices like motors and communication modules, that the main processor or microcontroller will interact with. Each peripheral must be correctly connected, and its communication protocols firmly established, ensuring seamless data exchange and control.
- **Creation of State Space**. The creation of state space is a fundamental step that involves defining all possible states that the system or agent can occupy within its environment. Identifying obstacles forms an integral part of the process. These obstacles, which may vary in nature and dynamics, create constraints within which the path planning must efficiently navigate. This obstacle identification is achieved through the LiDAR readings, which are converted to state spaces.

- **Neighbor Calculation on Each State**. This involves analyzing the connectivity and accessibility between states based on the system's movement rules and environmental constraints. For each state, the algorithm computes a set of neighboring states, which are the potential next steps the entity can take. This calculation considers several factors such as distance, direction, possible obstacles, and specific movement capabilities of the entity.[16]
- **Concurrently choose the best neighbor for each state**. This method involves evaluating all adjacent states or neighbors for each state in the system simultaneously, rather than sequentially. By leveraging parallel processing, the algorithm can assess multiple paths and options at once, significantly speeding up the decision-making process.

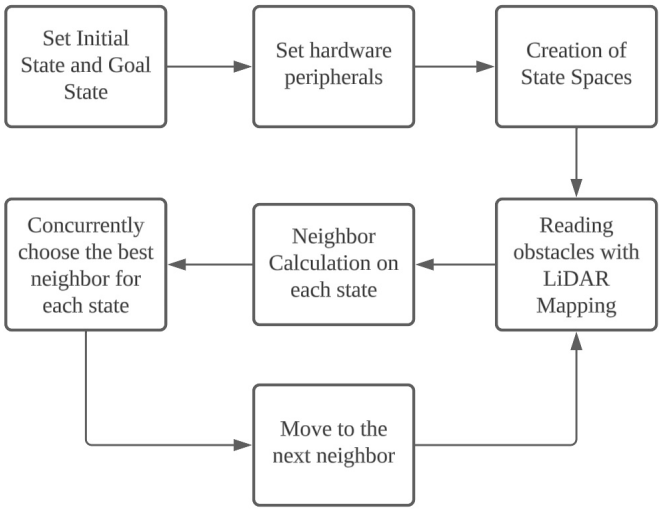


Figure 6: System block diagram.

4. RESULTS

This section presents the system integration, the mapping from LiDAR measures to state space, and the results obtained.

4.1 System integration

The system integration for testing the algorithm in a real omnidirectional robot moving through a physical environment consisted of the following aspects:

- Build a robust and working omnidirectional drive robot capable of sensing via a LiDAR and controlled via a microcomputer (Jetson Nano).
- Build each virtual map in real life representing the grid with black; giving a unitary gap between vertexes of 30 cm between each other and representing the obstacles with boxes.
- For every test of the project the initial coordinate is (2,2) and the goal coordinate is (7,7).
- Test each of the two maps ten times with the same starting and goal coordinates to measure the planning (or re-planning) times and the execution times for further analysis.
- Implement the algorithm in the robot's computer (Nvidia Jetson Nano) and verify the hardware.
- Analyze the behavior of the robot in static environments when the map is unknown.

Figure 7: Omnidirectional robot used for the implementation.

Fig. 7 presents a layered arrangement of various robotic components, each distinct in its function and appearance. On the top, there is a LiDAR and Jetson Nano, notable for its compact black casing, serving as the brain of the setup with its powerful computing capabilities. In the middle layer, we find an array of components: a Stepper Motors Driver, easily identifiable by its purple color, which plays a crucial role in controlling the precision of motor movements; alongside there are Antennas, represented as black cylinders, crucial for wireless communication; and a Robot Handler, comprising white semi-ellipses, which is likely involved in manipulating or interfacing with other parts of the robot.

On the bottom, the setup is completed with Mecanum wheels, known for their unique ability to move in any direction, which adds a versatile range of motion to the robotic assembly. This layered configuration illustrates a sophisticated blend of electronic and mechanical engineering, each component working in harmony to create a versatile and functional robotic system.

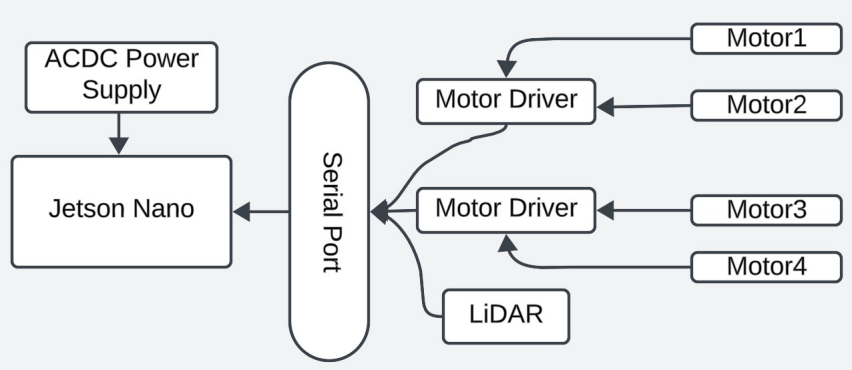


Figure 8: Connections between the electronic components involved in the robot.

Fig. 8 shows the connections and elements involved in having a working robot capable of moving around. The robot is capable of processing complex information (thanks to the Jetson Nano), capable of sensing by a LiDAR, and capable of controlling actuators (stepper motors) thanks to the Arduino nano microcontroller that acts as a motor driver. There is a voltage regulator to provide adequate current and voltage to each component.

4.2 From LiDAR measures to state space

The RPLiDAR sensor executes a comprehensive 360-degree scan, capturing measurements at each degree interval. This process results in a distance measurement, denoted as d, recorded in meters for each of the 360 degrees.

As illustrated in Fig. 9, the RPLiDAR collects this distance data for every angle θ, forming a complete circular profile of the surrounding environment.

To utilize this distance data for further analysis and applications, it is often necessary to convert these measurements into a Cartesian coordinate system, referred to as state space. This transformation involves decomposing the distance d measured at each angle θ into its corresponding x and y components. The conversion from polar to Cartesian coordinates can be achieved using the trigonometric relationships shown in Eq. 1, 2, 3, and 4.

$$\sin(\theta) = \frac{y}{d} \tag{1}$$

$$y = d\sin(\theta) \tag{2}$$

$$\cos(\theta) = \frac{x}{d} \tag{3}$$

$$x = d\cos(\theta) \tag{4}$$

Finally, to adapt the measurements for practical use in state space, the calculated distances must be normalized by the size of the state space grid, which in this case is 30 cm (0.3 m). This normalization process ensures that the coordinates fit appropriately within the defined grid size. Thus, the x and y coordinates for each distance d at its respective angle θ will be as shown in Eq. 5 and 6.

$$x = \frac{d}{0.3}\cos(\theta) \tag{5}$$

$$y = \frac{d}{0.3}\sin(\theta) \tag{6}$$

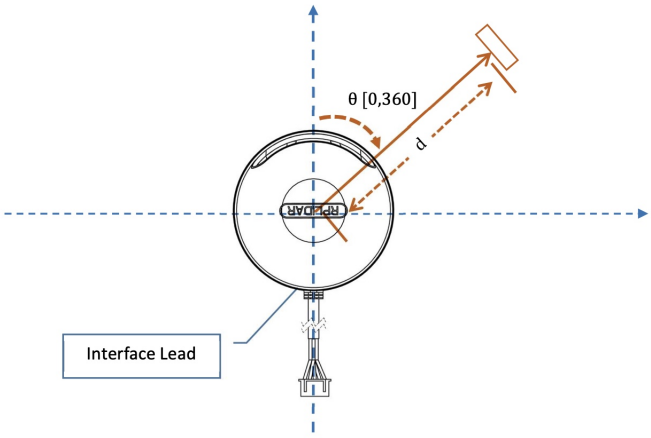
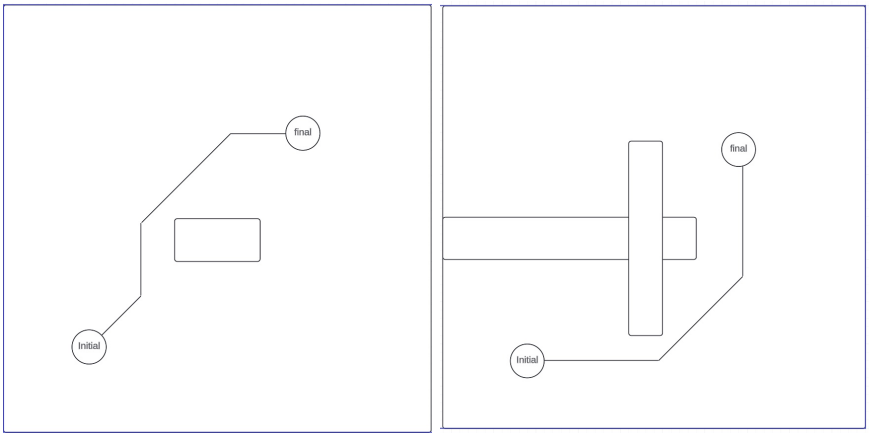


Figure 9: Visualization of LiDAR mapping. Image by Slamtec.

4.3 Physical results and performance

Fig. 10 shows the maps tested on a real environment with the following path. Fig. 11 shows the elements involved in the visualization of the motion of the omnidirectional drive robot through the map in a virtual environment performed in a Python program.



(a) (b)

Figure 10: (a) Map1, (b) Map2

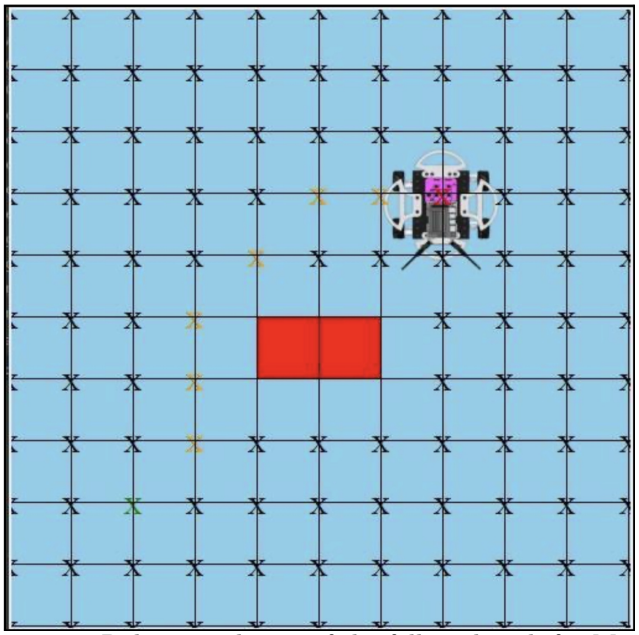


Figure 11: Robot simulation of the followed path for Map1.

The two different map configurations were designed and chosen to prove the reliability of the path planning algorithm and obstacle detection LiDAR-based mechanism, this allows a solid foundation on how to evaluate the experiment. The performance criteria for evaluation was the time it took the robot to move from one state to another. In Map1 the following path consists of 8 state spaces, beginning from (2,2) and ending at (7,7). Map2 has the same start and end as Map1 but the follow path consists of 9 visited state spaces.

Table 1: Results in seconds (s) regarding Map1.

State Space	Path Planning time (s)	Translation to next state time (s)	Total time (s)
1	1.63	1.22	2.85
2	1.55	1.15	2.70
3	1.57	1.18	2.75
4	1.59	1.21	2.8
5	1.61	1.24	2.85
6	1.68	1.16	2.84
7	1.70	1.18	2.88
8 (goal)	-	-	-
Average	1.61	1.19	2.81
Standard Deviation	0.055	0.030	0.059

Table 1 shows the planning and execution times for the mobile robot to achieve its goal of moving from an initial coordinate to a goal coordinate on Map1 with unknown obstacles for the robot; the map is the same as the one shown in Fig. 10a for the whole motion of the robot, but obstacles are unknown until the mapping is performed. As shown in Table 1, the average path planning time is 1.61s and that includes the LiDAR mapping and path calculation. On the other side, the average translation to the next state time is 1.19s, giving a total average time of 2.81s of planning and movement.

Table 2: Results in seconds (s) regarding Map2.

State Space	Path Planning time(s)	Translation to next state time (s)	Total time (s)
1	1.51	1.14	2.65
2	1.59	1.17	2.76
3	1.60	1.13	2.73
4	1.68	1.27	2.95
5	1.72	1.31	3.03
6	1.70	1.06	2.76
7	1.66	1.15	2.81
8	1.64	1.20	2.84
9 (goal)	-	-	-
Average	1.63	1.17	2.81
Standard Deviation	0.064	0.074	0.115

Table 2 aims to summarize the performance of the mobile robot in Map2, similar to Map1 but a bit different. In this table is notable that Path Planning time increases slightly when the obstacles queue contains more states. In addition, the Translation to the next state time increases when a diagonal movement is performed.

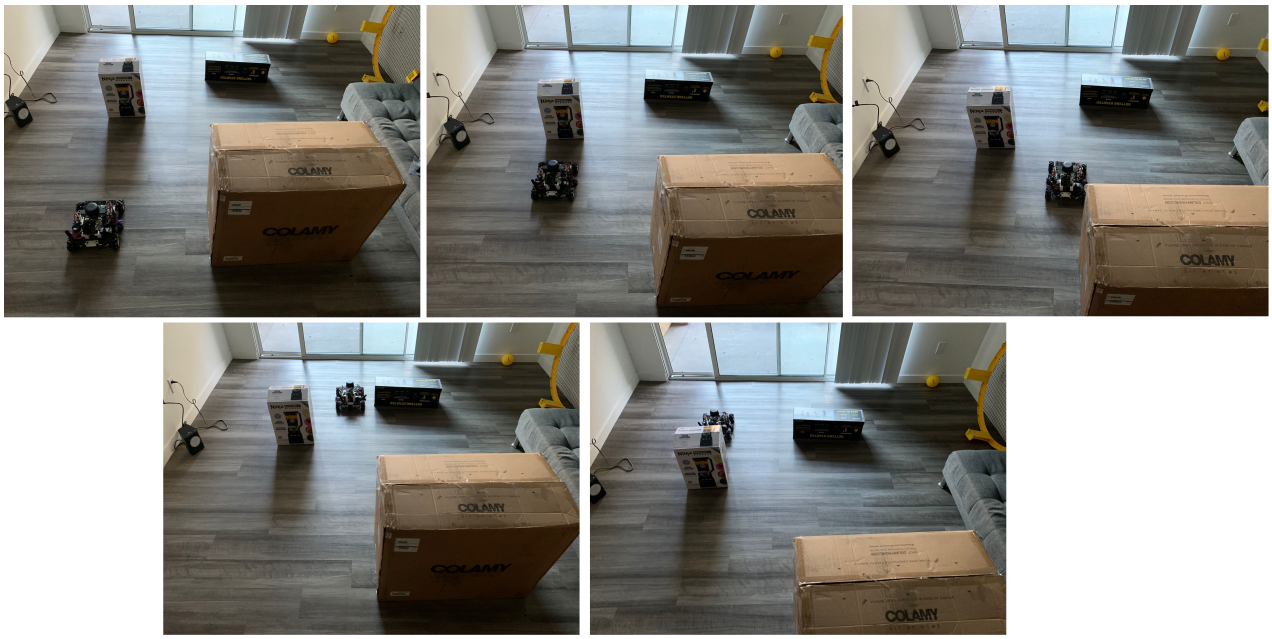


Figure 12: Real-life implementation on a mobile robot.

Fig. 12 shows some images of the final execution of the algorithm in a real mobile robot in an environment simulated with some cardboard as obstacles. It is even possible to appreciate that the robot encounters an obstacle and re-plans a new path until it arrives at the goal.

5. CONCLUSIONS

The work's primary goal, which was to implement iterative depth for a mobile agent operating within a state space using CUDA C/C++, has been accomplished. The utilization of parallel programming on the CUDA-enabled device has enabled the efficient execution of the algorithm, although with some minor margins of error that have not hindered overall functionality. This work serves as a compelling demonstration of the feasibility of applying data parallelism techniques to control and robotics projects. The use of CUDA libraries has proven instrumental in achieving parallelism within the source code, displaying the potential for leveraging GPU computing to enhance the performance and capabilities of systems in the realm of control and robotics. This achievement highlights the promising future of parallel programming in pushing the boundaries of computational efficiency in various domains.

However, there are still several areas that require enhancement for the system to work efficiently. The communication between the LiDAR readings and the main program currently takes approximately 1.5 seconds, necessitating improvements in communication time. Additionally, the motors used are quite heavy and power-hungry, indicating a need to consider lighter motors to reduce energy consumption and overall weight. The omnidirectional movement proves somewhat costly with the mechanical components currently in use, suggesting that enhancements in the mechanical design are warranted. Physical testing of the algorithm on 10x10 grids showed slightly better navigation times compared to a CPU; however, for much larger grids, the improvement in execution time with a GPU becomes significantly more pronounced. Furthermore, reducing the size of the subspaces could lead to smoother and more precise navigation for the robot. Addressing these details will be crucial in optimizing the system's overall performance and efficiency.

Finally, this work underscores the growing importance of parallel programming and GPU acceleration in addressing complex computational challenges. With the continuous advancement of hardware technology, such as GPUs, the potential for leveraging parallelism to tackle intricate problems across various fields continues to expand. The successful application of CUDA C/C++ in this context not only demonstrates its versatility but also encourages further exploration of parallel computing solutions for even more ambitious projects in control,

robotics, and beyond. As parallel programming techniques continue to evolve and mature, they hold the promise of unlocking new horizons in terms of computational power and efficiency, driving innovation and breakthroughs in numerous domains.

ACKNOWLEDGMENTS

This work was supported by the Coordinación Institucional de Investigación of CETYS Universidad, and by the Mexican National Council of Science and Technology (Consejo Nacional de Humanidades, Ciencias y Tecnologías, CONAHCYT).

REFERENCES

[1] Hwu, W.-m. W. and Kirk, D. B., [*Programming Massively Parallel Processors*], Morgan Kaufmann (2010).

[2] Olvera, T., Orozco-Rosas, U., and Picos, K., "Mapping and navigation in an unknown environment using LiDAR for mobile service robots," in [*Optics and Photonics for Information Processing XIV*], Awwal, A. A. S., Iftekharuddin, K. M., Diaz-Ramirez, V. H., and Márquez, A., eds., **11509**, 1150905, International Society for Optics and Photonics, SPIE (2020).

[3] Macias, L. R., Aleman-Gallegos, J. E., Orozco-Rosas, U., and Picos, K., "Map based localization using an RGB-D camera and a 2D LiDAR for autonomous mobile robot navigation," in [*Optics and Photonics for Information Processing XVII*], Iftekharuddin, K. M., Awwal, A. A. S., and Diaz-Ramirez, V. H., eds., **12673**, 126730F, International Society for Optics and Photonics, SPIE (2023).

[4] Orozco-Rosas, U., Picos, K., and Montiel, O., "Acceleration of path planning computation based on evolutionary artificial potential field for non-static environments," in [*Intuitionistic and Type-2 Fuzzy Logic Enhancements in Neural and Optimization Algorithms: Theory and Applications*], *Studies in Computational Intelligence* **862**, Springer (2020).

[5] Orozco-Rosas, U., Picos, K., Montiel, O., and Castillo, O., "Environment recognition for path generation in autonomous mobile robots," in [*Hybrid Intelligent Systems in Control, Pattern Recognition and Medicine*], *Studies in Computational Intelligence* **827**, Springer (2020).

[6] Orozco-Rosas, U., Montiel, O., and Sepúlveda, R., "An optimized GPU implementation for a path planning algorithm based on parallel pseudo-bacterial potential field," in [*Nature-Inspired Design of Hybrid Intelligent Systems*], *Studies in Computational Intelligence* **667**, 353–364, Springer (2017).

[7] Sezer, V. and Gokasan, M., "A novel obstacle avoidance algorithm: Follow the gap method," *Robotics and Autonomous Systems* **60**, 1123–1134 (2012).

[8] Kowalczyk, W., Przybyla, M., and Kozlowski, K., "Set-point control of mobile robot with obstacle detection and avoidance using navigation function - experimental verification," *Journal of Intelligent & Robotic Systems* **85**, 539–552 (2017).

[9] LaValle, S. M., [*Planning Algorithms*], Cambridge University Press, New York, NY, USA (2006).

[10] Orozco-Rosas, U., Montiel, O., and Sepúlveda, R., "Parallel evolutionary artificial potential field for path planning—an implementation on GPU," in [*Design of Intelligent Systems Based on Fuzzy Logic, Neural Networks and Nature-Inspired Optimization*], *Studies in Computational Intelligence* **601**, Springer (2015).

[11] Trobec, R., Vajtersic, M., and Zinterhof, P., "Parallel computing: Numerics, applications, and trends," in [*Parallel Computing: Numerics, Applications, and Trends*], 471–510, Springer (2009).

[12] Khatib, O., "Real-time obstacle avoidance for manipulators and mobile robots," in [*Proceedings of the 1985 IEEE International Conference on Robotics and Automation*], 500–505, IEEE, St. Louis, MO, USA (1985).

[13] Slamtec, *RPLIDAR A2M8 Development Kit Datasheet.* Slamtec (2018).

[14] Li, Y. and Olson, E. B., "Extracting general-purpose features from LIDAR data," in [*2010 IEEE International Conference on Robotics and Automation*], 1388–1393 (2010).

[15] Catapang, A. N. and Ramos, M., "Obstacle detection using a 2D LIDAR system for an autonomous vehicle," in [*2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*], 441–445 (2016).

[16] Ge, S. S. and Cui, Y. J., "New potential functions for mobile robot path planning," *IEEE Transactions on Robotics and Automation* **16**, 615–620 (Oct 2000).