Multilayer occupancy grid for obstacle avoidance in an autonomous ground vehicle using RGB-D camera

1st Jhair S. Gallego
Department of Mechanical and Mechatronics Engineering
Universidad Nacional de Colombia
Bogotá, Colombia
jhgallego@unal.edu.co

2nd Ricardo E. Ramirez
Department of Mechanical and Mechatronics Engineering
Universidad Nacional de Colombia
Bogotá, Colombia
reramirezh@unal.edu.co

***Abstract*— This work describes the process of integrating a depth camera into the navigation system of a self-driving ground vehicle (SDV) and the implementation of a multilayer costmap that enhances the vehicle's obstacle identification process by expanding its two-dimensional field of view, based on 2D LIDAR, to a three-dimensional perception system using an RGB-D camera. This approach lays the foundation for a robust vision-based navigation and obstacle detection system. A theoretical review is presented and implementation results are discussed for future work.**

***Index Terms*—Occupancy grid, ROS, RGB-D depth camera, Docker, self-driving ground vehicle (SDV), obstacle avoidance.**

I. Introduction

A. Problem definition

With the development of new technologies within the context of Industry 4.0, there is a push for intelligent and collaborative communication between robots in a factory. The main objective of this approach is for robots to be flexible in adapting their operations according to the changing demands of the factory. The implementation of collaborative robotics allows these agents to interact with one another, promoting flexibility in their operations. Adaptability in the manufacturing environment is not limited to robot interaction; it also addresses the need to adjust factory operations according to market dynamics. In a context where technologies are emerging rapidly and production needs are evolving, it is crucial for each robot to be flexible within the production chain. This flexibility facilitates an agile response to new demands without the need to incorporate new robots in every case.

The logistics of raw materials and manufactured items play a fundamental role in production processes. In the context of an experimental factory, there is an urgent need to transport items between the various machines that make up the facility. Self-Driving ground Vehicles (SDVs) enable the transportation of these items within production plants. These vehicles, upon receiving instructions from an external agent, plan trajectories and navigate autonomously, overcoming dynamic obstacles such as the presence of people or other robots. The use of a 2D LiDAR sensor is common in the perception system of these vehicles. However, in certain scenarios, the effectiveness of trajectory execution is impacted due to the limited field of view provided by the sensor. The 2D LiDAR-based perception system cannot detect obstacles whose position or geometry lies outside the sensor's field of view. As shown in Figure 1, any obstacle whose geometry does not intersect with the red plane will affect the effective generation of trajectories that allow for evasion, as the perception system will not account for it, resulting in a collision. This generates operational failures in the production processes of the factories where the vehicles operate. To address this limitation, we propose the integration of a depth camera to the perception system of these vehicles, extending the field of view to three dimensions and accounting for new obstacle detection improvements. In this work, we describe the process of integration of a depth camera (RGB-D) into the navigation system of an SDV, we present a theoretical review of the concepts around this integration, describe the implementation of a multilayer occupancy grid, and evaluate its effectiveness in strengthening the obstacle detection process.

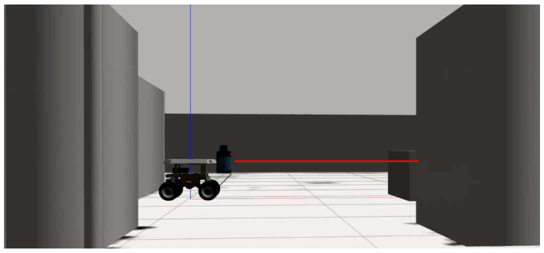


Fig. 1: Vertical field of view concept for LiDAR 2D.

B. Working vehicle

For the development of this work, the SDV II [1] from the Experimental Factory Laboratory (LabFabEx) at the Universidad Nacional de Colombia was used. The vehicle is shown in Figure 2.

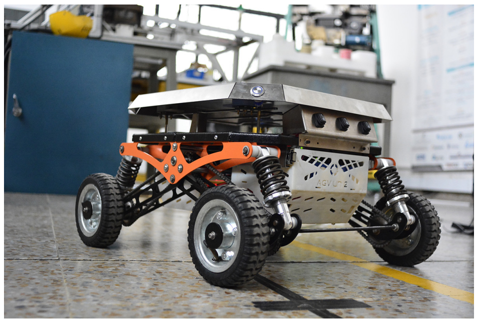


Fig. 2: SDV II vehicle from the LabFabEx [2].

This vehicle features a skid-steering kinematic architecture (see fig. 3), that allows it to move with two degrees of freedom. i.e., in the vehicle's local coordinate frame with origin in the geometric center of the vehicle, movement forward and backwards (x direction) and rotations around the ground plane (yaw) [3]. The vehicle's electronic components are classified as follows:

1) *Computing and Navigation System:* Composed of an Intel NUC mini-computer with 8 GB of RAM, Two USB 2.0 ports, and one USB 3.0 port [4]. The operating system used is Linux 18.04 (kernel v5.5), and navigation is implemented under the ROS Noetic framework [5].
2) *Perception System:* Uses the 2D LiDAR sensor Sick-LMS102 [6].
3) *Low-Level Control:* Uses a Tiva C Series board [7]. It facilitates serial communication with the motor drivers responsible for the vehicle's propulsion.

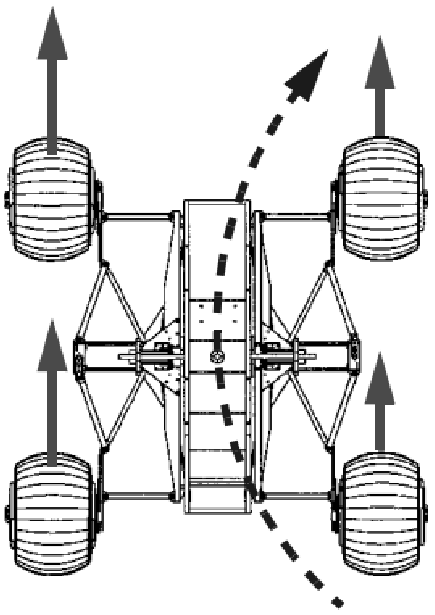


Fig. 3: Skid-steering architecture [8].

II. Background

This section presents the structure and functioning of specific components of the navigation system (*nav_stack*) that enable trajectory planning and execution based on data provided by the perception system. The study of the multilayer costmap, used in trajectory generation, is introduced. Finally, the concepts of the operating theory and optical properties of the RGB-D camera are also presented.

A. Navigation System

The autonomous navigation system is implemented using the *move_base* navigation package from ROS [9]. It uses costmaps to represent information about the environment's occupancy by objects. This information is used by local and global planners (*global_planner* and *local_planner*) for safe trajectory planning and generation. In the context of ROS's autonomous navigation stack, there are two main levels of costmaps: the *global_costmap* and the *local_costmap* (see fig. 4).

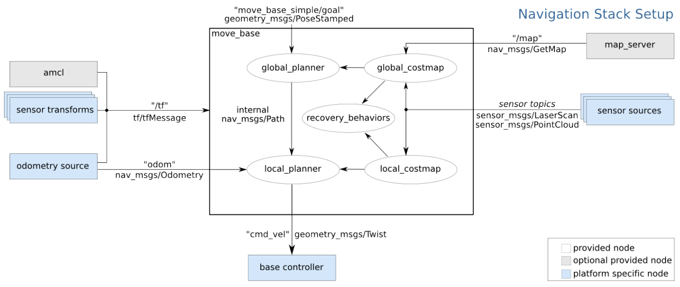


Fig. 4: Costmap under the navigation system architecture of the SDV [9].

Global Costmap: The global_costmap aims to provide a high-level occupancy representation of the environment and is used for global route planning. This costmap is generated using both static and dynamic information. The former generally comes from previously created occupancy maps and represents the unchanging environment, such as stationary robots or walls. On the other hand, the dynamic information provides a real-time representation of the robot's environment, such as a moving person or another vehicle, and comes directly from the perception system.

Local Costmap: The local_costmap, in contrast, describes the immediate environment of the robot and is used for local navigation. For example, it identifies nearby and moving obstacles. This costmap is updated more frequently and also utilizes information provided by the perception system.

Occupancy Grid: The main representation of the environment in the global and local costmaps is an occupancy grid. In this grid, each cell represents a region of the environment and has a value indicating the level of occupancy. Typically, a scale of discrete values is used, such as: 0 (free cell), indicating that the cell is unoccupied, or 100 (occupied cell), indicating the presence of an obstacle. This grid is what the trajectory planning system ultimately uses to generate safe routes for navigating the environment.

B. *Multilayer Costmap*

The overlay of the global and local costmaps is referred to as the Multilayer Costmap. The occupancy grid generated from this map integrates both global and local information. Figure 5 shows a conceptual representation of this map, where each costmap, whether global or local, can originate from specific sensors, such as LiDAR or cameras. Additionally, it is possible to use multiple costmaps from various sensors of the same type, such as several cameras in different positions, thereby strengthening the perception system.

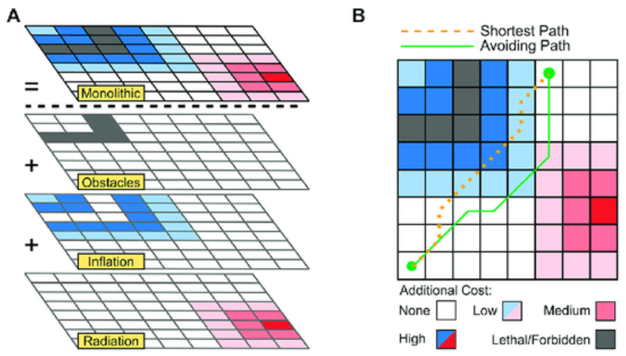


Fig. 5: The Multilayer Costmap concept [10].

C. *Optical Properties of the RGB-D Camera*

Field of View: The field of view (FOV) of a camera allows us to understand the extent of the observable scene in an image or video. It represents the angular range that the camera lens can capture. It is generally defined in both horizontal and vertical dimensions. The horizontal field of view determines the left-to-right range, while the vertical field of view covers the up-and-down range. Spatially, the FOV is described by a frustum, which is a geometric shape that plays a crucial role in understanding and visualizing the camera's field of view (see fig. 6). The frustum defines the volume of space that the camera can capture in a scene and is essentially a truncated pyramid, where the apex of the pyramid is at the camera lens, and the base of the pyramid represents the image plane or sensor.

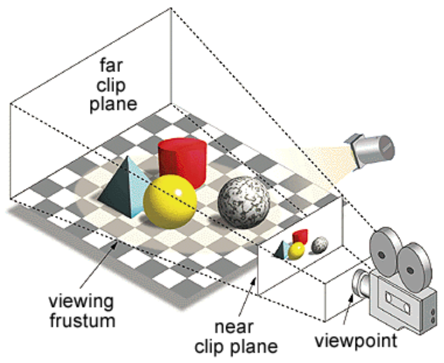


Fig. 6: Three-dimensional representation of a camera's field of view [11].

The camera's frustum is defined by three parameters: the near and far clipping planes, the horizontal and vertical fields of view, and the aspect ratio of the image. The near and far clipping planes represent the minimum and maximum distances from the camera at which objects will be in focus and included in the captured image. The shape of the frustum is determined by extending lines from the camera lens through the corners of the near and far clipping planes (see fig. 6). Similarly, the values of the horizontal and vertical FOV (*fovy* in fig. 7), often specified in degrees or radians, indicate the angular extent covered by the lens. Furthermore, the aspect ratio of the image, which represents the proportional relationship between width and height, influences how the FOV is distributed (see fig. 7).

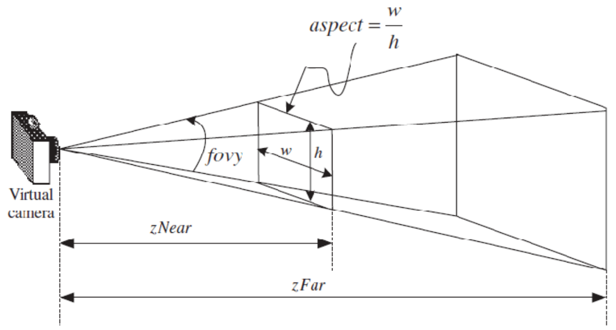


Fig. 7: Characteristic parameters of a camera's field of view[12].

Depth Map: The depth map or point cloud is a three-dimensional representation of the environment captured by a depth camera. In this context, depth is measured as the distance from the camera to objects in the scene. The point cloud consists of a set of coordinates (x, y, z), where each point represents the spatial position of a detected object in the image captured by the camera.
This type of three-dimensional information is fundamental for perception and autonomous navigation applications in robotics. In the context of ROS, the autonomous navigation stack uses the point cloud to build and update occupancy maps of the robot's environment. These maps are used to plan safe routes and avoid obstacles during autonomous navigation. This type of data is represented using the PointCloud2 message from the ROS framework [13]. The message' structure contains detailed information about the point cloud, including the three-dimensional coordinates of each point, as well as additional information such as data on color and signal intensity.

III. Implementation

A. ***Development Environment***

To avoid overloading the operating system on the NUC when running graphical applications on Linux, the distributed architecture of ROS is used under a client-server model (see fig. 8). In this way, the NUC can operate without a graphical interface (*headless* mode), and the data published by the topics are consulted over the Wi-Fi network. Visualization and data analysis applications run on a client, a computer connected to the same network, leveraging the resources of both devices. As shown in fig. 9, using the Rviz visualizer from an external computer, we can observe sensors' readings and how it is interpreted by the vehicle. This method facilitates the troubleshooting process and effectively monitors the vehicle's status in real-time. Additionally, this architecture allows multiple clients to query the vehicle's data, improving the scalability of the development environment.

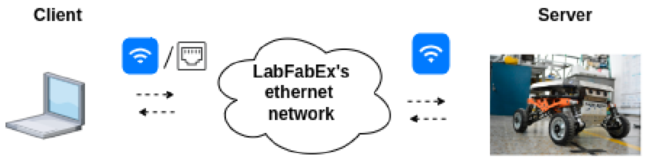


Fig. 8: Client-server architecture.

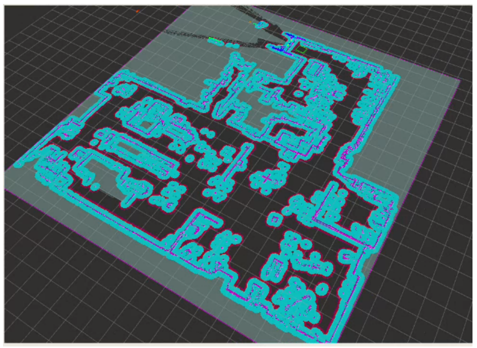


Fig. 9: Perception of the surrounding world by the SDV.

Docker Technology

To ensure consistency in software versions across different clients and to guarantee the replicability of the development environment, Docker is used. This technology allows the creation of a container defined in a plain text file, commonly called a Dockerfile, which (for the purposes of this work) unifies the use of Rviz, Gazebo, and ROS (Noetic) in a standardized environment, facilitating the replication of the development environment on any client.

B. *Intel RealSense D435i Depth Camera*

As a RGB-D camera, the Intel D435i reference is used. This camera offers two main advantages: it can be connected via a USB 2.0 port and has long-term technological support from the manufacturer. For integration with ROS, the installation process recommended by the manufacturer was followed as in [14]. Figure 10 illustrates the point cloud (in white) and the *stereo* image generated by the camera.

Optical Properties of D435i

The technical specifications of the camera relevant to this work are summarized in Table I.

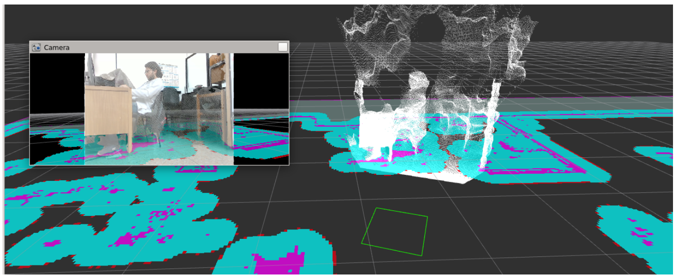


Fig. 10: Point cloud generated by the D435i camera at LabFabEx.

TABLE I: D435i FOV Specifications (adapted from [15]).

Operating Range	0.3m to 3m
Field of View (FOV)	87° (vertical) 58° (horizontal)
Aspect Ratio	16:9
Depth Resolution	1280 x 720

Support Design for Mounting the Camera on the Vehicle

The navigation system of the SDV requires knowing the *pose* of the camera in order to perform the coordinate transformation to the central coordinate system of the SDV. This is achieved by providing the location of the camera's geometric center with respect to the central coordinate system (see fig. 11). With this, it is possible to properly interpret and process the generated point cloud. Therefore, it is necessary to know the *pose* of the camera to perform the corresponding transformation (see fig. 11).

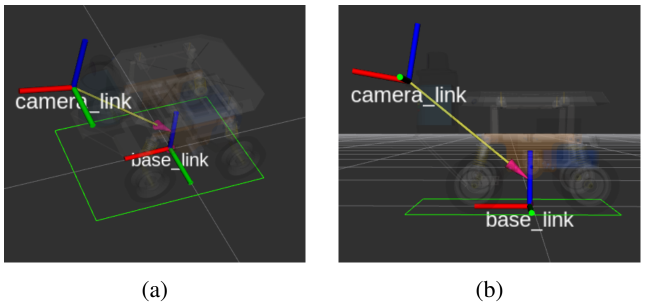


Fig. 11: Relationship between the *base_link* and *camera_link* coordinate systems.

Since it is not necessary for the camera to perform translations or rotations about its own axis, but rather to remain fixed, it is important that the position of the camera relative to the SDV base is always the same. To this end, a support was designed and manufactured to allow the camera to be directly coupled to a pre-existing mechanism on the SDV. This way, the position vector (and hence the coordinate system transformation) of the camera with respect to the base (TF_{cb}) can be obtained (see fig. 11). The position of the *camera_link* coordinate system relative to the *base_link* coordinate system is described by the vector 1:

$$\mathbf{TF}_{cb} = \langle 0.345, 0, 0.28 \rangle m \tag{1}$$

The final design is shown in fig. 12d, and the integration with the pre-existing mechanism along with the mounting of the camera is shown in fig. 12.

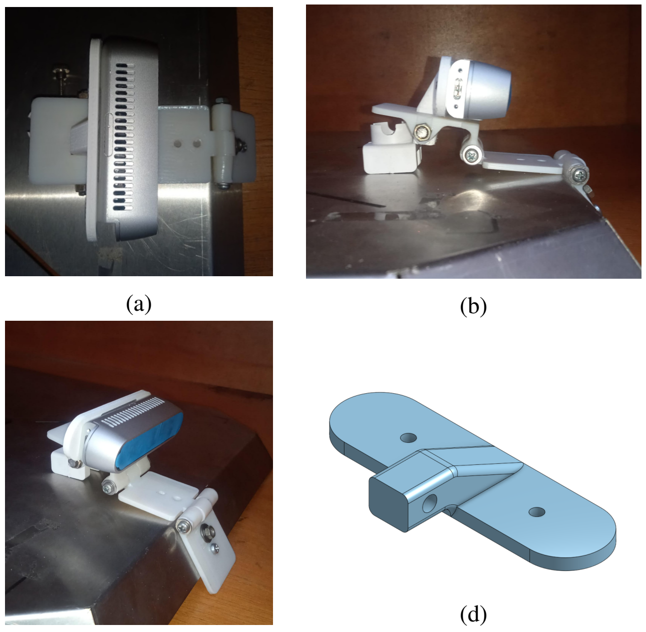


Fig. 12: D435i camera support. *(a), (b), (c)*: Camera support mounted on mechanism. *(d)*: Isometric view of support's CAD model.

C. Integration with the Navigation System

As mentioned in sections II-A and II-B, integrating a new sensor requires adjusting certain configuration parameters in the navigation system. In Figure 13, it can be observed that as the viewing plane of the camera's frustum extends further, the points exhibit greater dispersion and fail to accurately represent objects at those distances, such as walls in this case. Consequently, the costmap generates a false positive for an obstacle (represented in purple), which impacts trajectory planning for target points intersecting the affected areas. Therefore, the detection range of points as obstacles is limited to a frontal distance of less than $2m$ and a height range between $0.35m$ and $1.0m$. Table II provides a summary of these and other relevant configuration parameters. These define a frustum as described in section II-C and the ROS topic's name and data type to which the point cloud information comes from. The functionality of each parameter can be referenced from the official documentation of the /move_base package [16].

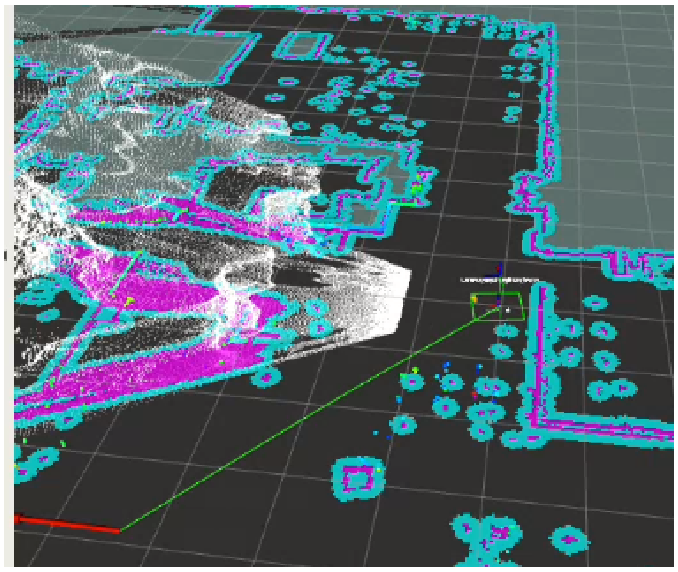


Fig. 13: False positive clasification of obstacles when using default detection range limits in *costmap_2d* parameters.

TABLE II: Camera integration parameters for costmap

data_type	PointCloud2
topic	camera/depth/color/points
marking	true
clearing	true
max_obstacle_height	1.0
min_obstacle_height	0.35
obstacle_range	2.0
raytrace_range	2.0

IV. Results

To verify that the camera is indeed assisting in trajectory planning by identifying objects, the following scenario was implemented. An obstacle in the form of a bridge, represented by plastic (hereafter referred to as *the bridge*), was placed at a height of $60cm$, in such a way that it could not be detected by the LiDAR but could be identified by the camera, due to its vertical field of view (see fig. 14). The test consists of providing a target position to the vehicle, such that it initially generates a trajectory that should pass under the bridge.

In figure 15, it can be seen that initially, a trajectory is generated so that the vehicle should go under the bridge. Once the camera identifies the obstacle, it is effectively projected onto the costmap, highlighted by a red box in figure 16a. Similarly, an alternative trajectory (green line) is recalculated to reach the proposed target location. In figure 16b, from the camera's perspective, both the bridge (obstacle) and its projection generated by the navigation system can be observed.

V. Conclusions

- The integration of the RGB-D camera significantly improves the autonomous vehicle's ability to identify and avoid obstacles, especially those not visible by the LiDAR. This provides robustness in trajectory planning in dynamic environments.
- The use of technologies like Docker to standardize the development environment and ensure experiment replicability is a key point of this work. This allows for efficient integration and testing of new cameras and algorithms, facilitating the maintenance and expansion of the autonomous navigation system in the future.

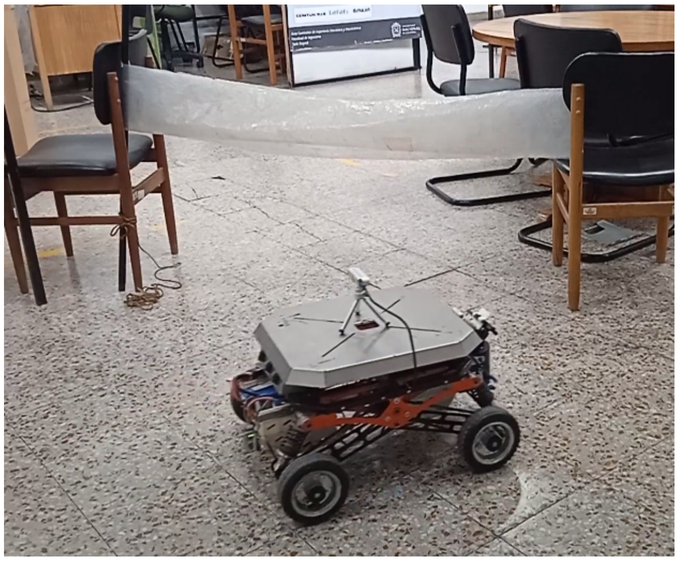
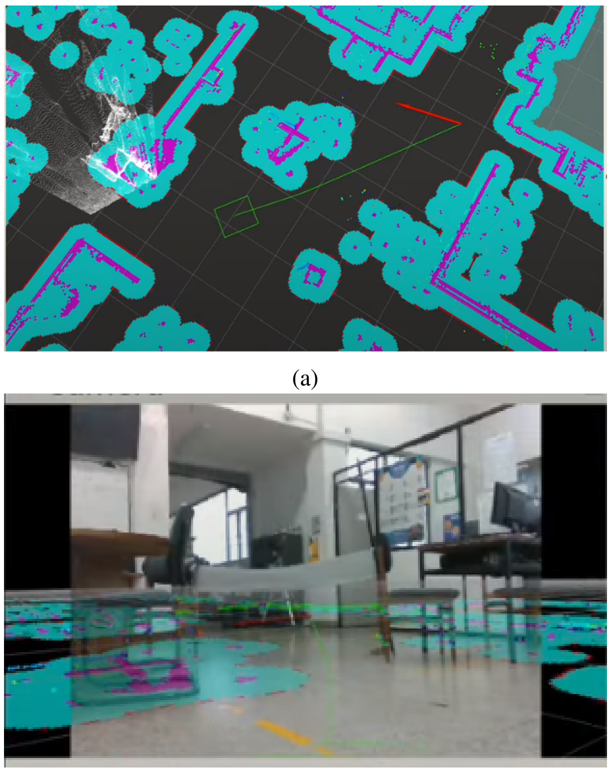


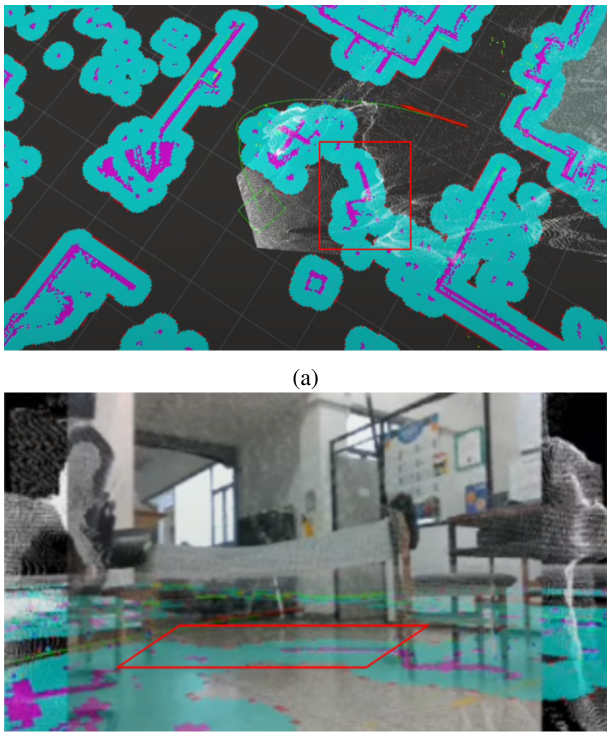
Fig. 14: Testing scenario for obstacle avoidance with the RGB-D camera.



(a)

(b)

Fig. 15: Testing scenario before dynamic obstacle avoidance.



(a)

(b)

Fig. 16: Testing scenario after dynamic obstacle avoidance.

- The integration of the depth camera opens opportunities for the development of new algorithms based on machine vision, which can enhance the vehicle's operational capabilities, such as implementing decision trees based on object identification.

VI. Future work

Filtering depth points by adjusting the camera's frustum parameters is a simplistic approach. The multilayer costmap can be enhanced by using more sophisticated algorithms, such as density-based point cloud filtering. The current method fails to identify and filter out some single-point data generated either because of camera's noise, or when facing reflective surfaces such as glass. Implementing a density-based filter could potentially mitigate this issue, taking into account that objects have a high point cloud density as can be seen in figure 13.

References

[1] J. Galvis Sarria, J. Mesa Rodríguez, J. Chávez Chávez, J. Pineda Pinzón, E. Córdoba Nieto, and C. Velásquez Hernández, "Diseño, desarrollo y construcción de vehículos guiados automáticamente (agv) e implementación de funciones de navegación mediante sensores láser," Universidad Autónoma de Occidente, 2016.

[2] Unimedios. Plataforma robótica móvil consolida desarrollo de la industria inteligente. [Online]. Available: https://agenciadenoticias.unal.edu.co/detalle/plataforma-robotica-movil-consolida-desarrollo-de-la-industria-inteligente

[3] T. Wang, Y. Wu, J. Liang, C. Han, J. Chen, and Q. Zhao, "Analysis and experimental kinematics of a skid-steering wheeled robot based on a laser scanner sensor," *Sensors*, vol. 15, no. 5, pp. 9681–9702, 2015. [Online]. Available: https://www.mdpi.com/1424-8220/15/5/9681

[4] I. Corporation, "Intel nuc kits," 2023. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/nuc/kits/products.html

[5] Ros Community, "Ros noetic official documentation," 2020. [Online]. Available: http://wiki.ros.org/noetic

[6] S. AG, "Productos y soluciones," https://www.sick.com/es/es/c/g91901, 2024.

[7] T. Instrumentos. Tiva™ tm4c123gh6pm microcontroller datasheet. [Online]. Available: https://www.ti.com/lit/ds/spms376e/spms376e.pdf?ts=1701452414847&ref_url=https%253A%252F%252Fwww.ti.com%252Ftool%252FEK-TM4C123GXL

[8] B. Shamah, "Experimental comparison of skid steering vs. explicit steering for a wheeled mobile robot," p. 14, 1999, https://www.ri.cmu.edu/pub_files/pub1/shamah_benjamin_1999_1/shamah_benjamin_1999_1.pdf.

[9] Ros Community, "Move base package arquitecture," Year. [Online]. Available: http://wiki.ros.org/move_base

[10] A. West, T. Wright, I. Tsitsimpelis, K. Groves, M. Joyce, and B. Lennox, "Real-time avoidance of ionising radiation using layered costmaps for mobile robots," *Frontiers in Robotics and AI*, vol. 9, p. 862067, 03 2022.

[11] pcmag, "Frustum," 2023. [Online]. Available: https://i.pcmag.com/imagery/encyclopedia-terms/viewing-frustum-_frustum.fit_lim.size_1050x.gif

[12] "Frustum fov," Year. [Online]. Available: https://www.researchgate.net/figure/A-viewing-frustum-defined-in-OpenGL-to-emulate-the-real-camera_fig1_261264621

[13] Pointcloud2 ros msg structure. [Online]. Available: https://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/PointCloud2.html

[14] IntelRealSense, "Intel realsense ros wrapper," Year. [Online]. Available: https://github.com/IntelRealSense/realsense-ros#installation

[15] I. Corporation, "Intel realsense depth camera d435i," 2023. [Online]. Available: https://www.intelrealsense.com/depth-camera-d435i/

[16] Ros 1 Community, "Costmap 2d sensor management parameters." [Online]. Available: https://wiki.ros.org/costmap_2d/hydro/obstacles