Path planning with far-away obstacles detection under uncertainty

Mantoani, L. M.[a], Pérez-del-Pulgar, C. J.[a,*], Luque, G.[b]

[a] *Universidad de Málaga, Department of Systems Engineering and Automation, Space Robotics Laboratory, Málaga, Spain*
[b] *Universidad de Málaga, ITIS Software, Málaga, Spain*

Abstract

Hoy en día se están investigando robots de exploración terrestre y espacial *más rápidos* en respuesta a la creciente demanda de capacidades de exploración e investigación más rápidas, eficaces y rentables. Para estas plataformas móviles rápidas la identificación y evasión de *obstáculos lejanos* son críticas, ya que su alta velocidad implica la necesidad de tener en cuenta el mayor número posible de obstáculos cercanos y lejanos para el cálculo de la trayectoria global, evitando cualquier posible accidente debido a su velocidad y al tiempo de cálculo de los algoritmos de replanificación. Debido a su distancia, los obstáculos lejanos no se incluyen en los mapas locales, que están limitados por el alcance de las cámaras de profundidad. Por estas razones, este artículo propone el uso de técnicas de Inteligencia Artificial para detectarlos a partir de imágenes y estimar sus tamaños y posiciones con un cierto grado de incertidumbre. Los obstáculos detectados se incluirán posteriormente en los mapas globales, corrigiendo la trayectoria global en caso de colisionar con ellos.

Keywords: Sistemas robóticos autónomos, Guiado navegación y control, Robótica inteligente, Robots móviles, Percepción y detección.

1. Introduction

Planetary exploration missions are becoming more and more important since space exploration encourages international cooperation and innovation, bringing us closer to know whether there is life on another planet and satisfying the human desire to explore the universe and to understand the world around us. Furthermore, the exploration of other planets and especially of the moon is particularly important, since it allows us to gain scientific knowledge about the formation of the solar system, as long as the evolution of the planets and of the moon itself.

On the other hand, terrestrial mobile robots allow us to explore and investigate areas on Earth that are difficult or dangerous for humans to access. These robots are becoming increasingly important in a wide range of fields, including manufacturing, logistics, healthcare, agriculture and exploration. They are designed to move on land, either autonomously or under human control, and can perform a variety of tasks that are difficult or dangerous for humans to do.

Autonomous navigation is essential to succeed in future planetary exploration missions with rovers (space exploration robots) to extend the traversed distance and maximize the number of places visited during the mission, taking into account that real-time communication between Earth and other planets such as Mars is quite impossible (Bajracharya et al., 2008). Autonomy is very important for terrestrial mobile robots too, since it enables them to operate independently in a variety of environments, such as hazardous or inaccessible areas where human intervention may not be possible or safe. Due to the importance of autonomous navigation, both in space and terrestrial applications, the implementation of a *Guidance, Navigation and Control* (GNC) architecture is essential (Azkarate et al., 2020). The Guidance component is the one in charge of planning a path for the mobile robot to follow, ensuring a safe traverse, avoiding any possible obstacle that can be found on its way. The Navigation component is responsible for the perception of possible obstacles that can affect the mobile robot traverse, and the robot localisation, using GNSS on Earth and visual odometry or SLAM techniques on remote planets. Finally, the control component is the one in charge of ensuring the robot is correctly following the generated path, providing actuator commands based on the mobile robot kinematics (Mantoani et al., 2022).

Taking into consideration that most of the mobile robots should perform their tasks in a limited time and traversing the longest distances possible, their velocity is key, allowing to reach further places in less time. For this reason there is an increased interest in developing faster mobile platforms.For example, VIPER is a rapid lunar rover designed by NASA to search ice and other potential resources on the lunar surface (Colaprete et al., 2019). Nevertheless most of the currently existing mobile platforms are not very fast: planetary exploration rovers, for example, are usually moving at less than 10 cm/s,

*Corresponding author: carlosperez@uma.es

e.g. the Perseverance rover from NASA, which moves on the martian surface around 4.2 cm/s. This is due to mechanical and security problems, as long as several issues related to the local level of autonomy, since the mobile robot should be able to generate and follow a path, and rapidly detect and avoid as many obstacles as possible. In the case of fast robots, this is not a trivial task, therefore, autonomy needs to be improved through advancing the GNC system.

In the case of the Navigation subsystem, apart from the local obstacles that can be detected in a local 3D map generated from the exteroceptive sensors, i.e. LIDAR, stereocameras, depth cameras, etc. (Jaspers et al., 2017), it is crucial to take into consideration further obstacles when the rover is moving faster, i.e. the ones that are not visible in the local 3D map but that are present in the images taken by the robot cameras. These types of obstacles can be considered as *far obstacles*, since they do not require to perform an urgent avoidance manoeuvre.

Therefore, this paper proposes a modification in the GNC architecture to take into consideration far obstacles. For this purpose, the navigation component has been extended to detect obstacles on monocular images using AI. Later on, these obstacle are fed to the guidance component to be taken into consideration during the path planning process. The proposed path planner is based on the *Fast Marching Method* (FMM) (Sethian, 1999), which is a numerical algorithm that is able to generate the optimal and smooth path based on a cost map. The method to place these far obstacles into the cost map is the key contribution of this paper. It uses certain parameters from the vision system and the information provided by the obstacle detection AI based algorithm.

2. Methodology

The limited range of the traditional depth cameras implies the inability to include distant obstacles within local 3D maps, limiting the performance of autonomous detection and navigation systems. However, thanks to Artificial Intelligence techniques it is possible to analyze complex images and extract relevant features to identify distant obstacles, even if the uncertainty related to visual information poses an additional challenge, since it is difficult to define the exact position of a distant obstacle from a single image. Given the two-dimensional nature of images, the depth and three-dimensional location of objects can be difficult to determine, but taking into account the size of the local 3D maps and the intrinsic parameters of the depth cameras it is possible to identify a region in which the obstacle is likely to be found, estimating its size taking into account a certain degree of uncertainty. The detection of this type of obstacles is very important, since the faster the robot is, the safer should be the path, and the sooner should any possible obstacle been taken into account: waiting for the robot to get nearer to the obstacle in order to have it included in the local map could be too late to avoid it.

The purpose of this work is to use Artificial intelligence techniques to detect distant obstacles in the images, investigating how to obtain a first approximation of their size, representing them by means of triangles whose apex would be located at the furthest valid point of the local map, in the same direction as the relative far obstacle. The apex location has been chosen taking into account that the only information available on the rocks is that they are outside of the map, but not their distance from its limits, and taking into account a certain degree of uncertainty. These obstacles will be added to the global maps, correcting the global path in case it collides with them.

The main responsible of the guidance component is the Path Planner (Sánchez-Ibánez et al., 2019), that generates a path to reach a final pose based on a global cost map. Firstly a map of the scenario in form of a global *Digital Elevation Map* (DEM) is required, and it is analyzed at the beginning of the planning algorithm to check its quality. After analyzing it, the robot initial position and the goal position are required to start the algorithm. Then, the global DEM is processed, in order to differentiate between safe and obstacle areas. Hence, a *traversability map* 1 is generated, that, based on the information provided by the DEM, assigns a value to each map node in order to classify them into different areas, taking into account their morphological characteristics.

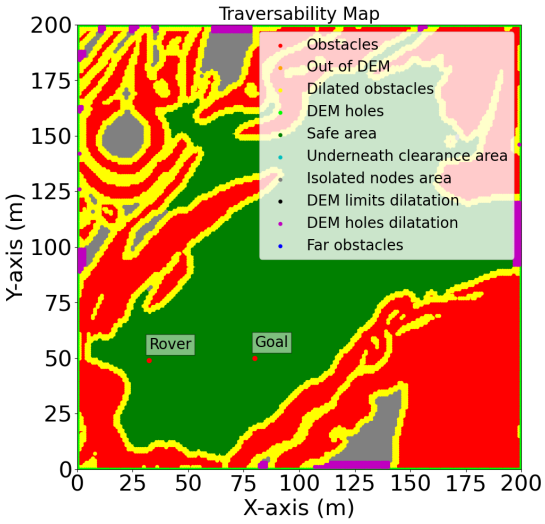


Figure 1: Example of global traversability map

In Figure 1 the green areas are the valid nodes, that can be safely traversed by the rover. The red areas are the obstacles, that are dilated (yellow nodes) according to the rover size and a safe distance, ensuring the rover will not enter on this area. Grey areas are the isolated nodes, valid nodes that cannot be reached by the rover since they are surrounded by obstacles. Finally, light green areas are unknown, since they are not clearly visible, and for this reason they are classified as DEM holes, and are also dilated for the robot security (magenta areas).

From the previous traversability map, a *cost map* is computed, as can be seen in Figure 2.

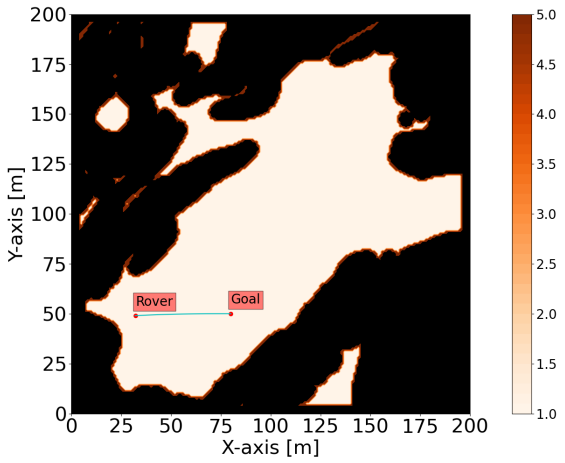


Figure 2: Example of global cost map

The cost map is generated by assigning a cost to each map node representing the amount of effort, i.e. time or resources required to move through it. The costs go from 1, that are the safer nodes, to 5, the most dangerous nodes, while the obstacles and the unknown areas are set to *infinity*, meaning that they should be avoided in order to ensure the robot safety. In Figure 2 the white nodes are the safer ones, with a cost of 1, while the black nodes are the risky areas that shouldn't be traversed, having a cost set to infinity. Obviously the higher values are set to the nodes that are closer to the forbidden areas, and these values decrease while getting closer to the valid nodes, gradually, from infinity to 1: this transition is represented with different shades of orange.

The cost map is used by path planning algorithms to determine the optimal path between two points, considering the cost of traversing different areas. Normally the nodes chosen by the path planning algorithm to compute a safe trajectory are the ones whose cost map value is closer to 1.0, or with the lower possible value.

Once the cost map has been obtained, a global path is calculated taking into account the global cost map computed, the goal position and the robot initial position. The trajectory is obtained by means of the Fast Marching Method (FMM) (Gómez et al., 2019), a numerical scheme that is used to calculate the solution to non-linear equations called Eikonal, and that can be applied to path planning algorithms to expand a wave over the cost map in order to extract a trajectory from it. The FMM is also used to check the safety of the previously computed global path each time a local DEM is obtained from a camera, correcting it in case it collides with obstacles (Sánchez-Ibánez et al., 2019).

Each time a new path is provided, it is sent to the *Control* component (Gerdes et al., 2020), which generates linear and angular velocity commands that are later translated to wheels speeds commands, to be understandable by the robot.

Several images and DEMs have been collected to implement the proposed far obstacles detection and avoidance, including several rocks at different distances from the robot. All the images are later processed using the YOLOv5 Neural Network, which has been able to recognize all the rocks within the images, providing information regarding the perception success rate and the coordinates of the bounding box surrounding the detected rocks, which is later used to estimate the far obstacles position. The X coordinate of the central pixel of each bounding box is used to compute the angle between the far obstacle and the camera pose (that is the current robot pose) taking into account the camera intrinsic parameters, such as its Horizontal Field of View (HFOV) and its width (1):

$$\alpha = (2x_p - c_w)(\frac{c_f}{2c_w}), \tag{1}$$

where α is the horizontal angle between the far obstacle and the camera, x_p the X coordinate of the central pixel of the bounding box provided by the neural network, c_w the camera width in pixels and c_f the horizontal field of view of the camera in degrees.

After obtaining the angles for each detected rock, the local validity map is used to identify the furthest valid point of the map in the direction of each rock, in order to obtain its coordinates and use them as the apex of the triangle that will be drawn to simulate the relative obstacle. It is important to remark that all detected rocks will be taken into account and processed, the near and the far ones, since it is not possible to correctly classify which ones are near obstacles and which are the far ones, and in terms of security it is better to consider all of them, instead of possibly discarding the wrong ones. When finally all the obstacles have been correctly identified, the coordinates of their apex, their success rate and their angle are used to estimate their shape for the global maps.

The rocks are represented in the global DEM as triangular obstacles, due to the camera *cone of vision*, also known as the *visual field*: it is the portion of the environment that a camera can see at any given moment, which shape is similar to that of an isosceles triangle, with the apex of the cone located at the lens and the base extending outward into the environment. The reason for choosing a triangular shape is simple: when viewing an object in perspective, the size of the object for the camera depends on both its actual size and its distance from the lens. Objects that are closer will appear larger to the lens than objects that are farther away, even if they are actually the same size. This means that if a camera sees an object at a given distance with the same shape and size as another object that is further away, the furthest one is the biggest, since if it is moved at the same distance as the nearest one it would be bigger. Therefore, for the same shape and size, the further an object is, the bigger it is.

The base of each triangle (b) is proportional to the width of the bounding box (m) provided by the YOLOv5 Neural Network, $b = k_{pm}m$, where k_{pm} is the conversion factor to translate the width of the bounding box from pixels to metres, while the height of the triangle (h) is equal to the maximum distance the system can detect a far obstacles (r), $h = r$.

A function has been codified in order to identify all the points within the triangles: its inputs are the apex coordinates, the height, the base and the angle defining the obstacle direction. The first step is the computation of the position of the

base vertices (2) (3):

$$v_1(x,y) = (cos(\alpha)(x-k)+sen(\alpha)j, -sen(\alpha)(x-k)+cos(\alpha)j) \quad (2)$$

$$v_2(x,y) = (cos(\alpha)(x+k)+sen(\alpha)j, -sen(\alpha)(x+k)+cos(\alpha)j) \quad (3)$$

where v_1 and v_2 are the triangle base vertices, α is the horizontal angle previously computed, x is the X coordinate of the triangle apex, $k = \frac{b}{2}$, half of the triangle base, and $j = y + h$, sum of the Y coordinate of the triangle apex and the height of the triangle.

Once the position of the base vertices is computed, all the points of the triangle sizes are identified, and all the nodes included between these sizes are collected. Hence, the relative global traversability map nodes are classified as FAR_OBSTACLE, a new traversability classification created for these specific types of obstacles.

After including these new obstacles in the global traversability map, the global cost map is also updated. In this case the success rate of the detected obstacle is required to define the cost of each triangle node: the higher the success rate, the higher the probability of having an obstacle, and therefore the higher the cost of that obstacle, to prevent the robot from approaching it. Obstacles with a success rate lower than 50% have been discarded, since such low values may result in a high number of false positives or unreliable results. An algorithm has been codified, that, taking into account the success rate of each obstacle detected, assigns to the relative triangle a proportional value going from 1, the lower value of the cost map (if the success rate is lower than 60%), to infinity (if it is 100%).

After having updated the global cost map, the Fast Matching Method is executed again, and the resulting path will take into account all new obstacles. Since the best trajectory is chosen taking into account the cost of the nodes, the ones free of any obstacles and with the lower cost will be chosen, being the best candidates. In the unlikely case there is no available path to connect the robot and the goal through completely safe nodes, the FMM would provide the path with the lowest cost, even if some of its nodes have a cost higher than 1.

Several experiments have been carried out to test the proposed algorithm, and they will be explained in the following section.

3. Results

In this section some of the experiments to test the proposed algorithm will be illustrated, starting from the software and hardware technologies employed to carry them out.

3.1. Experimental setup

This algorithm has been integrated into an already developed GNC architecture for ROS2. It is able to generate an optimal path to reach a goal position based on a global map, and repairing it each time a new obstacle is detected within the local DEMs, as long as to generate the robot commands, i.e. translational and rotational speeds, to be able to follow the generated path.

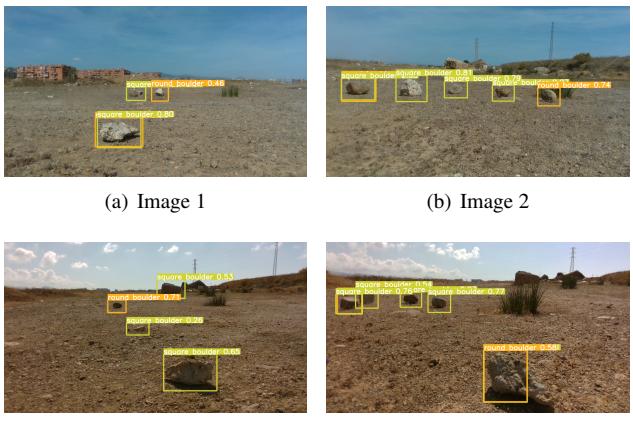
After updating the Guidance component, the far obstacles algorithm proposed in this paper has been integrated, adding new functions and topics as long as creating a new ROS2 node to read all the information provided by the YOLOv5 neural network. To execute and test the proposed algorithm, the CoppeliaSim simulator has been used, and a Lua file has been codified to subscribe and publish into several ROS2 topics, in order to connect to the simulator the Docker container with the Guidance component and the far obstacles algorithms.

To perform the tests real images were needed, showing near and far rocks in front of the robot. For this reason several images and DEMs have been collected using the OpenUMARov[1], a small rover that has been built through the years at the Space Robotics Lab at the University of Malaga. It's an open-source, low-cost mobile robotic platform that is usually used to test algorithms and acquire images and video. This rover has been equipped with an *Intel RealSense Depth Camera D435i*, an active stereoscopic depth camera that has been used to acquire images and local DEMs, with a range (r) of up to 10 meters, a horizontal field of view (c_f) of 87 degrees and a width (c_w) of 1280 pixels. The collected dataset can be found in Zenodo [2] public repository.

In the following section the performed experiments will be illustrated.

3.2. Algorithm execution

Several experiments have been carried out in order to test the proposed algorithm. This section focuses on one of them since it is considered the most relevant and representative of the system functioning.



(a) Image 1 (b) Image 2

(c) Image 3 (d) Image 4

Figure 3: YOLOv5 processed images

Given a global DEM, the first step is the computation of a global traversability and cost map from it, as long as a first

[1] https://github.com/spaceuma/OPEN-UMA-Rover
[2] https://doi.org/10.5281/zenodo.7890632

global path computed taking into account obstacles and safe areas. While the robot is moving local images and maps (previously collected using the RealSense camera) are periodically sent through ROS2 topics. These images are classified by the YOLOv5 neural network as shown in Figure 3, where several possible obstacles, including far and near rocks, have been correctly identified.

An example of a local DEM related to these images can be seen in Figure 4, where the red stars represent all the far obstacles detected.

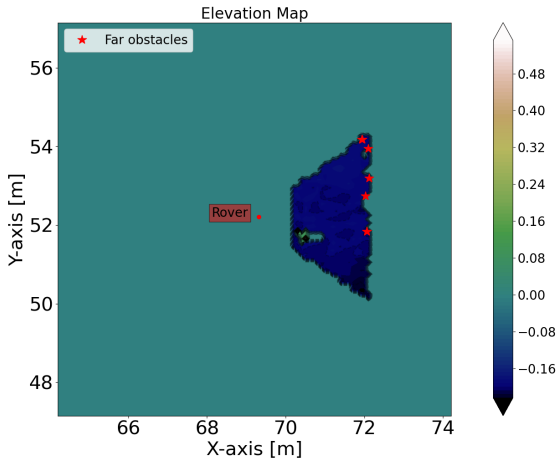


Figure 4: Example of the fourth image.

Once the algorithm has been executed and the far obstacles position has been correctly estimated, the global traversability map is updated, as long as the global cost map, where the global path is corrected in order to avoid the detected far obstacles.

While the rover is following the global trajectory previously corrected, new images and local maps are periodically sent through the ROS2 topics: each time a new detection is received the algorithm is executed, and all the maps are updated with the new far obstacles estimated, while the global path is also corrected to avoid them. A video[3] was published, showing a test carried out in simulation.

Figure 5 shows the cost maps from four algorithm executions. The path has been corrected 4 times: each time new far obstacles are identified the path is updated, making it safer for the robot to follow. The first cost map computed 5(a) shows the only valid far obstacle detected in Figure 3(a), since obstacles with a success rate lower than 50% have been discarded to avoid unreliable results, as was previously explained. The second cost map computed 5(b) shows how the trajectory is updated again taking into account the new far obstacles detected in Figure 3(b), that are represented with triangles with similar shades of orange, having quite similar success rates. The obstacles detected in Figure 3(c) can be seen in the third cost map computed 5(c), where the FMM chooses a path very close to the far obstacles, being the best candidate in terms of speed and security. Finally, the cost map is updated again 5(d) with the last far obstacles detected from Figure 3(d): in this case the triangles can be better identified, having different shades of orange due to their divergent success rates.

All these tests have been carried out in simulation, and it has been demonstrated that the robot is correctly able to follow each new updated path, the local ones as long as the global ones, reaching the final goal with no errors and with the correct position and orientation.

3.3. Processing times

Processing times are critical for fast mobile platforms, since they influence their ability to make quick and efficient decisions while moving fast. These times depend on the algorithm efficiency as long as on the hardware employed. The computer used to perform all the experiments has *3.40GHz Intel Core i7-6700* CPU, based on the x86_64 architecture and with 8 cores, as well as an *Nvidia GeForce GTX 1070 TI* graphics card with 8GB of dedicated memory and a RAM of 32GB.

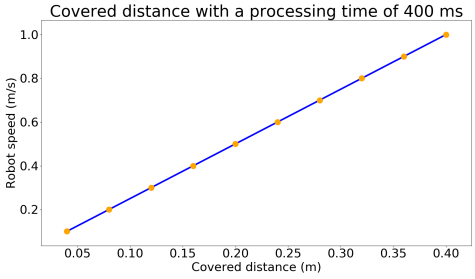


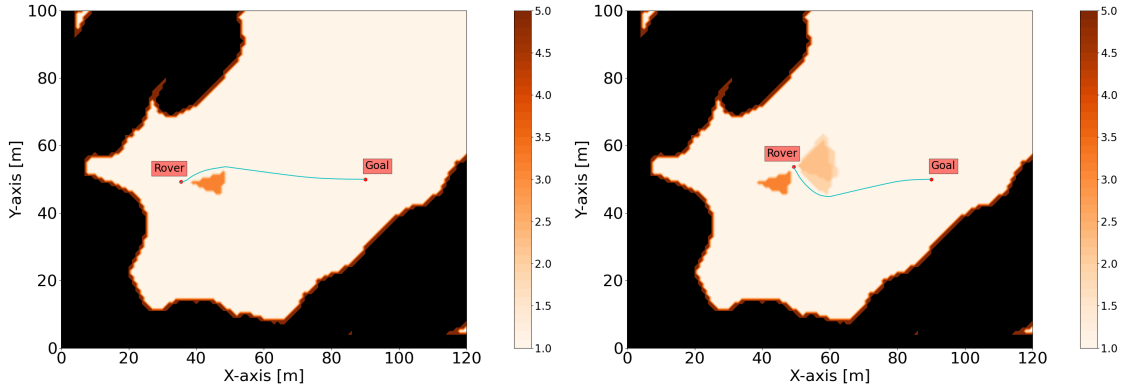
Figure 6: Covered distance depending on the robot's speed

To estimate the whole computation time firstly it is necessary to compute the image processing time, which is divided into the image acquisition (*8.5 ms*), the image pre-processing (*1.2 ms*), the image inference (*31.6 ms*) and the image post-processing (*1.3 ms*), with a total image processing time of about *42.6 ms*. The second step is the computation of the time required for the far obstacles algorithm, which includes the local maps processing (*184.129 ms*) and the global maps and path update (*164.488 ms*), giving a total computation time for the algorithm of *348.617 ms*. The sum of these values gives an average processing time of *391.217 ms*, less than 400 ms. Figure 6 shows the distance covered by a mobile platform during these computation steps depending on its speed, considering a total processing time of *400 ms*. Obviously, the faster the robot, the higher the covered distance, and the lower the robot security. Even at the highest speed of 1 m/s, the covered distance is quite small, giving to any fast robot enough time to carry out all necessary avoidance manoeuvre, with no risk of collision.

4. Conclusions

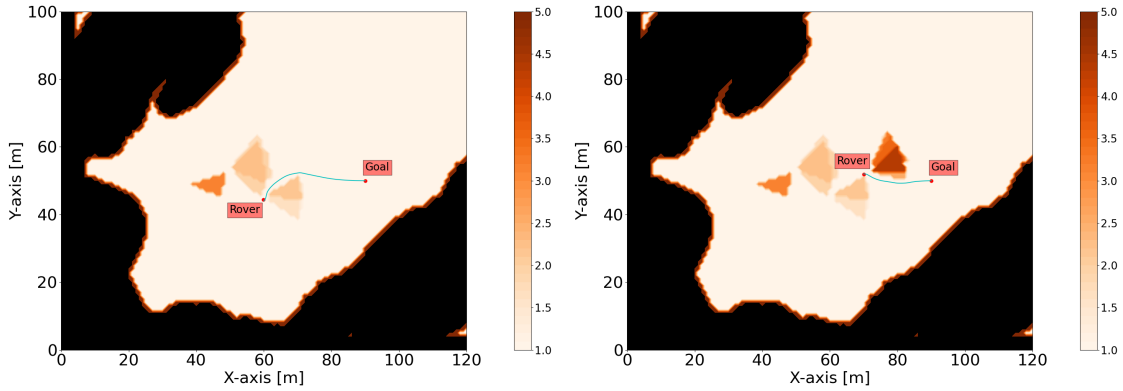
The aim of this work was to provide safer trajectories for rapid mobile platforms, applying the Fast Marching Method for

[3] https://youtu.be/7W-Xz7ach8c



(a) Planning of the first image

(b) Planning of the second image



(c) Planning of the third image

(d) Planning of the fourth image

Figure 5: Cost maps showing the algorithm execution

fast robots, using Artificial Intelligence techniques to detect far obstacles from images and estimating their shape and position, in order to update the global maps with this new information and to correct the global path in case it collides with them.

The inclusion of both nearer and further obstacles in the detection and mapping process significantly increases the security of the robot, ensuring that mobile platforms are not only aware of immediate obstacles but also of potential risks in the distance. This consideration is particularly important for rapid robots, since they need enough time to react and to correct their path in case it is necessary. Extending their awareness to distant obstacles brings them to plan their trajectories avoiding potentially hazardous situations, increasing their safety.

References

Azkarate, M., Gerdes, L., Joudrier, L., Pérez-del Pulgar, C. J., 2020. A gnc architecture for planetary rovers with autonomous navigation. In: 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp. 3003–3009.

Bajracharya, M., Maimone, M. W., Helmick, D., 2008. Autonomy for mars rovers: Past, present, and future. Computer 41 (12), 44–50. DOI: 10.1109/MC.2008.479

Colaprete, A., Andrews, D., Bluethmann, W., Elphic, R. C., Bussey, B., Trimble, J., Zacny, K., Captain, J. E., 2019. An overview of the volatiles investigating polar exploration rover (viper) mission. In: AGU Fall Meeting Abstracts. Vol. 2019. pp. P34B–03.

Gerdes, L., Azkarate, M., Sánchez-Ibáñez, J. R., Joudrier, L., Perez-del Pulgar, C. J., 2020. Efficient autonomous navigation for planetary rovers with limited resources. Journal of Field Robotics 37 (7), 1153–1170.

Gómez, J. V., Álvarez, D., Garrido, S., Moreno, L., 2019. Fast methods for eikonal equations: an experimental survey. IEEE Access 7, 39005–39029.

Jaspers, H., Himmelsbach, M., Wuensche, H.-J., 2017. Multi-modal local terrain maps from vision and lidar. In: 2017 IEEE Intelligent Vehicles Symposium (IV). IEEE, pp. 1119–1125.

Mantoani, L., Castilla-Arquillo, R., Paz Delgado, G. J., Perez-del Pulgar-Mancebo, C. J., Azkarate, M., et al., 2022. Samples detection and retrieval for a sample fetch rover.

Sánchez-Ibánez, J. R., Pérez-del Pulgar, C. J., Azkarate, M., Gerdes, L., García-Cerezo, A., 2019. Dynamic path planning for reconfigurable rovers using a multi-layered grid. Engineering Applications of Artificial Intelligence 86, 32–42.

Sethian, J. A., 1999. Fast marching methods. SIAM review 41 (2), 199–235.