UNIVERSIDADE D
COIMBRA

Afonso Elói Carvalho

Navigation, Planning and 3D Traversability Analysis in Forest Environments

Master's Dissertation in MIEEC, supervised by Doctor David B. S. Portugal and presented to the Faculty of Science and Technology of the University of Coimbra.

October 2021

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

Navigation, Planning and 3D Traversability Analysis in Forest Environments

Afonso Elói Carvalho

Coimbra, October 2021

UNIVERSIDADE D
COIMBRA

Navigation, Planning and 3D Traversability Analysis in Forest Environments

Supervisor:

Doctor David Bina Siassipour Portugal

Jury:

Prof. Doctor Paulo Mendes Breda Dias Coimbra

Prof. Doctor Cristiano Premebida

Doctor David Bina Siassipour Portugal

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

Coimbra, October 2021

Acknowledgements

To my supervisors, for overwhelmingly exceeding even my wildest expectations about what a truly exceptional guidance is, both scientific and personal. My special thanks to my supervisor, Professor David B. S. Portugal, for the incredible support, flexibility and friendliness shown while enforcing the highest standards;

To Gonçalo S. Martins, my co-supervisor, mentor and friend, who gave up countless hours of his time with the *a piori* knowledge that he would get no recognition for it (not officially anyway). For being one of the people who helped and inspired me the most in this short time;

To Professor João Filipe Ferreira, for all the enthusiasm, kindness and patience shown while leading fruitful discussions and providing scientific support well beyond his obligations;

To my parents and sister, who stood behind me doing their best to keep me from falling, and even a better job at getting me back up. Without you, this moment would not have happened;

To Pintas, one of the main characters in the story of my life, for the unconditional support and motivation;

To Camila Santos, my girlfriend and limitless source of support, whom with her mere presence pulls me out of my darkest moments. May our future be half as bright and happy as I envision it;

To D. Cristina and Sr. Artur, for all the support and kindness shown throughout this journey;

To Elvis Borges, my dearest friend, for all the deep philosophical discussions and all the support throughout many dimensions of my life. May our reunions never cease to happen;

To Filipe Barreto, for being the most thoughtful person I have ever met and for helping me in ways I cannot even begin to describe;

You have my eternal respect.

Resumo

A navegação autónoma em ambientes 3D não estruturados representa um grande desafio para a robótica moderna. Embora as florestas, em particular, sejam ambientes hostis que apresentam diversos tipos de obstáculos e perigos para robôs móveis autónomos, elas também fornecem informações valiosas e oportunidades para os mitigar.

Este trabalho consiste numa técnica inovadora de análise de travessia e planeamento de caminhos em 3D que processa mapas de nuvens de pontos tri-dimensionais para gerar informações de gradiente do terreno. Isto permite-lhe realizar uma análise estatística do ambiente percetível para que possa prever a irregularidade geral do terreno, bem como a presença de obstáculos. Esta informação permite gerar caminhos geralmente eficientes, pois evitam grandes declives quando existem caminhos viáveis menos exigentes, reduzindo implicitamente o desgaste do equipamento e os riscos associados.

Neste documento é apresentada uma revisão do estado da arte, identificando as principais lacunas de pesquisa e fornecendo uma contribuição científica com o desenvolvimento deste trabalho. A técnica proposta foi testada e comparada com outras em quatro cenários realistas da nossa autoria, simulados no exterior, e os resultados são apresentados e discutidos.

***Palavras-Chave*:** Robótica Florestal; Análise de Travessia; Navegação 3D; Planeamento de Caminhos; Veículo Terrestre Não-Tripulado.

Abstract

Autonomous navigation in unstructured 3D environments poses a great challenge for modern Robotics. Although forests, in particular, are harsh environments that present all kinds of obstacles and dangers for autonomous mobile robots, they also provide valuable information and opportunities to mitigate them.

This work presents a novel traversability analysis and path-planning technique that processes 3D pointcloud maps to generate terrain gradient information and perform a statistical analysis of the perceived environment so that it can predict the terrain's overall roughness, as well as the presence of obstacles. This information allows us to generate paths that are generally efficient, as they avoid major hills when more conservative paths are available, thus implicitly reducing the wear of the equipment and the associated risks.

We perform a review of the state of the art, identify key research gaps and provide a scientific contribution with the development of this work. The proposed technique has been tested and compared against other techniques on four realistic outdoor simulated scenarios of our own design, and the results are presented and discussed.

***Keywords*:** Forestry Robotics; Traversability Analysis; 3D Navigation; Path Planning; UGV.

"I will not follow where the path may lead, but I will go where there is no path, and I will leave a trail."

— Muriel Strode, *Wind-Wafted Wild Flowers*

Contents

List of Acronyms

ACO	Ant Colony Optimization
APF	Artificial Potential Field
GPS	Global Positioning System
IMU	Inertial Measurement Unit
IRI	International Roughness Index
LiDAR	Light Detection And Ranging
PCL	Point Cloud Library
RGB	Red Green Blue
RGB-D	Red Green Blue Depth
ROS	Robot Operating System
SEMFIRE	Safety, Exploration and Maintenance of Forests with the Integration of Ecological Robotics
SLAM	Simultaneous Localization and Mapping
ToF	Time-of-Flight
UAV	Unmanned Air Vehicle
UGV	Unmanned Ground Vehicle
VFF	Virtual Force Fields

List of Figures

List of Tables

1 Introduction

Navigation is a subject of vital importance in modern robotics, endowing all sorts of robots with the ability to move within their workspace, ranging from modern autonomous vacuum cleaners to the rovers used to explore uncharted celestial bodies.

Forest maintenance is both dangerous and labor-intensive, requiring staggering amounts of time, money and human resources, making it a prime candidate for automation. Such autonomy requires a reliable mean of unsupervised navigation, which is a sizeable challenge considering that forests are highly unstructured, dynamic outdoor environments. As such, these environments have gained increasing attention over the past few years, with many new research and development efforts being undertaken, tackling main issues such as autonomous maintenance and management in completely [2] and semi- [3, 4] unstructured forest environments, as well as more delicate environments including orchards [5], vineyards [6] and plantation fields [7, 8].

As such, this work proposes an innovative path planning technique that uses the concept of mechanical effort from [9] to enable the generation of paths that are less demanding for a Unmanned Ground Vehicle (UGV), especially avoiding steep sections of terrain, potentially improving the fuel economy and reducing the mechanical wear by minimizing the mechanical effort that the robot is subject to. We pursue our goal by using a two-dimensional costmap – which is a discrete grid of cells, where each cell represents a portion of the terrain that holds relevant topographical information – to represent the perceived environment, in which a traversability analysis and successive path planning are performed to generate said paths.

Although this work is not platform-specific, we aim to account for the specific limitations and capabilities of a differential heavy-duty forestry robot (Fig. 1.1) under

Figure 1.1: The SEMFIRE UGV considered throughout the course of this Dissertation work [1].

the Safety, Exploration and Maintenance of Forests with the Integration of Ecological Robotics (SEMFIRE) project. The main goal of the SEMFIRE project is to develop a fully autonomous system capable of undertaking large forest maintenance missions, with the support of an UGV of large dimensions and several Unmanned Air Vehicle (UAV). The path planning technique developed in this work is meant to empower with autonomous navigation capabilities the ground agent of the system.

The proposed technique for path planning and traversability analysis has been tested and compared with other techniques on a realistic 3D Gazebo [10] simulation environment that we have set up, with multiple scenarios with different characteristics and associated complexities, providing a solid basis for the comparisons that have been performed.

The remainder of the work is organized as follows: in Chapter 2, we present a state-of-the-art study on traversability analysis, path planning and navigation techniques. In Chapter 3, we describe in detail the proposed architecture and the simulation environment that we have developed. In Chapter 4, experimental setup and results are presented, detailing the different steps, such as chosen scenarios and performance metrics, finishing with a discussion of the results. Finally, Chapter 5 is

the conclusion of this dissertation, providing relevant avenues to explore for future work.

2 Background and Related Work

2.1 Fundamentals of Navigation and Path Planning

Navigation and path planning are key aspects when considering robot's motion. Considering that we deal with a differential UGV, it is important to know the basics of controlling a platform of this kind.

A **differential platform** has the ability to fully control the angular speed of both wheels independently. This means that curved paths (as seen in Fig. 2.1) are achieved by varying the ratio at which both wheels are turning, therefore not needing an additional steering degree. It is also important to notice that these platforms can rotate over their own axis, which empowers them to invert their direction of motion without needing additional space other than the one it is occupying in that very moment. Nevertheless, these platforms are **non-holonomic**, which means that they have less controllable degrees of freedom than total degrees of freedom in the task space, i.e. they cannot move to any direction without first needing to readjust their angle [11].

When it comes to **autonomous navigation**, we consider as fully autonomous a vehicle that can, without any sort of human intervention during its execution phase, navigate through its workspace without unintentionally colliding with existing obstacles (whether static or dynamic), using information collected from its onboad sensor(s).

Nowadays, there are a multitude of sensor modalities that robots use in order to achieve their autonomy, such as:

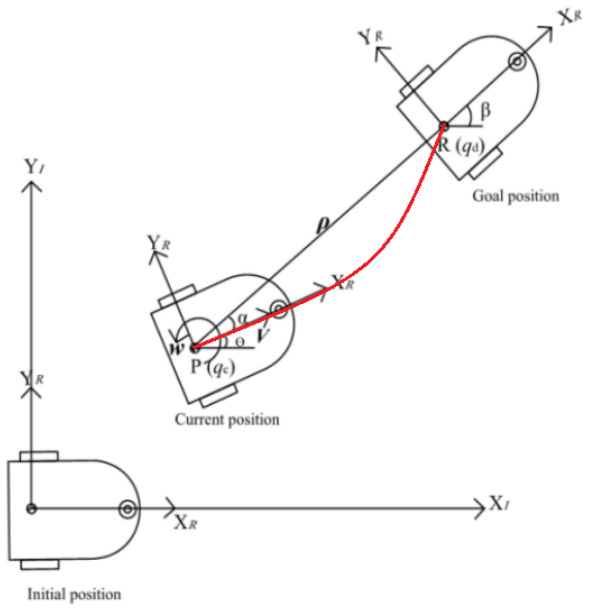


Figure 2.1: Typical behavior of a differential robot. Adapted from [11].

Light Detection And Ranging (LiDAR) - Works by emmiting beams of laser at a given time, waiting for the reflected signal to come back, and measuring the time that passed between those two events, being then able to calculate the distance that separates the device from the obstacle. This is mainly used in line of sight, but there are situations where LiDARs have been used to scout the grounds of, for instance, a forest from an unmanned air vehicle (UAV) where the only thing seen by a regular camera was foliage [12].

Time-of-Flight (ToF) cameras - Time of Flight cameras are similar in working principle to LiDAR (some of them actually operate with laser light, like LiDAR does), but may operate with different forms of artificial light sources. These cameras reconstruct scenes at a high rate, being able to be implemented in real-time systems.

Red Green Blue Depth (RGB-D) cameras - Similar to a common three channel Red Green Blue (RGB) camera, this devices captures colored scenes, with the differentiating ability of capturing depth information as well. These cameras generally use pulsed light or stereo vision for capturing depth in indoor environments, and usually stereo vision to capture depth in outdoor scenarios.

Capitalizing on sensor information, **odometry** is a technique that detects and quantifies position variation over time, consequently producing an estimation of the localization of the robot. Odometry data can be collected, for instance, from encoders positioned on the joints/wheels of a robot, called **wheel odometry**, or from a setup of cameras, in which case it is called **visual odometry**.

Another broadly used type of odometry estimation is called **LiDAR-Based Odometry**, where the detection of position variation of the robot over time and its consequent localization is achieved by detecting the position variation of environment features over time on simultaneous readings from the range sensor – called **scan matching** – mounted on a moving robot.

There are also other noteworthy types of odometry, such as **visual-inertial odometry**, where data from an image sensor and an Inertial Measurement Unit (IMU) is fused to generate the estimation of pose, and **RBG-D odometry** that also uses data fusion from an RGB camera and a depth sensor to achieve the same purpose.

Resorting to sensors, robots have the ability to detect obstacles up to a given distance and with a certain belief, which means that they can, for instance, navigate through an environment with a purely **reactive behavior**, i.e. they just drive straight towards a goal point, only changing their trajectory if an obstacle enters their detection radius. It is intuitive to understand that this way of navigating is not ideal, given that the robot has no understanding of the general environment it is in, increasing the chances of getting stuck in specific situations with no real way of solving the deadlock other than randomness or environment-specific analysis and so on.

An important concept in mobile robotics is **mapping**, which can be defined as the ability of an agent to construct a map of the environment using its sensors. Although the concept of mapping is independent from the concept of **localization** – which can be defined as the problem of determining the pose of a robot relative to a well-known frame of reference [13] – they are not completely inseparable in the sense that, unless the localization source is trivial (i.e. mapping from a stationary source),

it is not possible to map while in motion without having some sort of localization information.

Localization, however, may be performed without needing a map. Global Positioning System (GPS)-based localization, among others, is a common example of mapless localization. There are two main types of localization:

Relative Localization comes from relative methods such as odometry. These methods share the common premise that the pose estimation in instant `t` depends on the pose calculated in instance `t-1`, meaning that they cannot perform discrete corrections, thus accumulating error through time. Also, the localization they provide is typically centered in the reference frame of the sensor (or any other frame as long as the transformation between this frame and the base frame of the robot is known).

Absolute Localization is obtained by using methods such as GPS triangulation or trilateration referenced to a global frame. These methods do not dependent on the previous iterations of computation, which means that there is the possibility of performing discrete pose corrections, thus minimizing the inherent errors. This type of localization can, for instance, be used to correct the estimation of pose given by relative methods.

Usually, localization and mapping are implemented together, which is called Simultaneous Localization and Mapping (SLAM). This problem is solved by using the data obtained through the robot's sensors to fill the equivalent area on the map, and *simultaneously* locate both the robot and other landmarks. SLAM allows to generate the map and locate the robot in it concurrently, which makes it appropriate for a number of applications, including forest navigation [14].

In order to increase the robustness and capabilities of these robots, it is desirable to have a **map** of the surrounding environment. There are various types of maps, such as topological maps [15], feature maps, 3D maps and occupancy grid maps [16], being the latter the most relevant to this work. An occupancy grid map is a discrete representation of the environment, where the continuous real space is discretized into

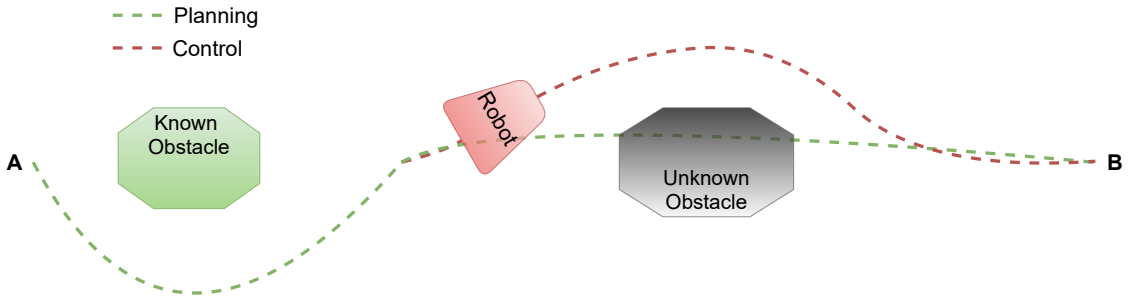


Figure 2.2: Illustration showing the planning and control phases.

cells of a given size, each containing a value that represents some characteristic of the terrain, for instance its occupancy state or traversability cost. Maps can consist of *a priori* full knowledge of the environment that is given to the robot, can consist of partial information that is known from the start, or can even be empty. In all the above cases, it is necessary to have some way of updating the map – this is useful even with a complete initial map if we are dealing with, for instance, unexpected obstacles – so that the robot's knowledge of the ambient is increasingly higher, as well as for self-localization purposes. That is why, as mentioned earlier, SLAM is commonly used for navigation, path planning and traversability analysis.

Having now established the importance of a map and the ability to create/update one during operation, we can introduce **path planning**. Path planning is an area within Robotics on its own, whose objective is to generate a feasible collision-free path from one place to another [17]. Although simpler or less demanding path planning tasks do not necessarily need an optimization strategy or even a mathematical representation, generally path planning problems are described as optimization problems, in which the objective is to find the optimal path on the given map that connects an initial pose to a goal one following a safe, efficient and collision-free trajectory [18]. Path planning can be divided into two sub-groups: planning and control.

Planning refers to the action of using the information of the map, if one exists, as well as the sensor readings in order to generate a global path that links the start and goal poses, even if the goal is not in the known horizon of the robot. Thus, the planning phase can sometimes be associated with the concept of **global planning**.

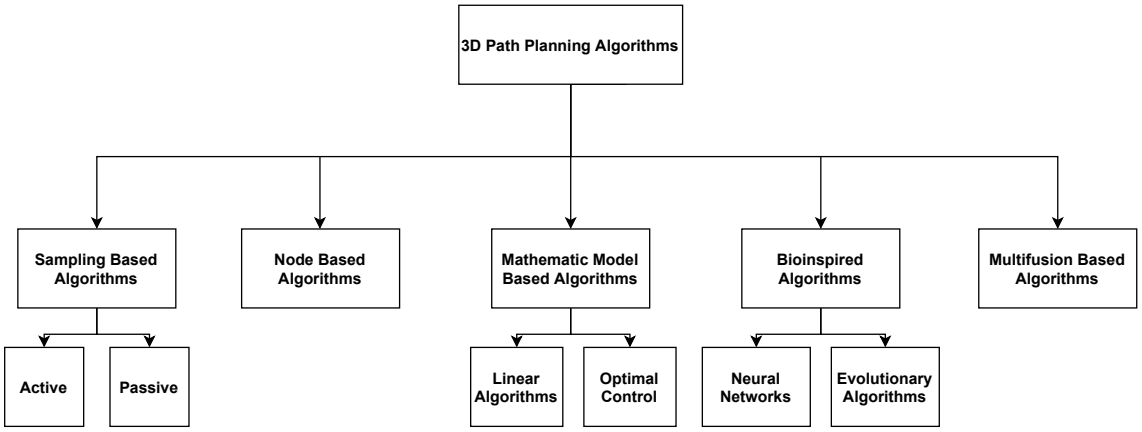


Figure 2.3: Illustration of the 3D path planning taxonomy, inspired in [18]

Control designates the action of following the path generated in the planning phase. In this phase the robot is limited to its sensory horizon, generally planning a small distance ahead and trying to closely follow the obtained path, deviating from it whenever necessary in order to guarantee a collision-free path. Thus, the control phase can sometimes be associated with the concept of **local planning**.

Considering the advantages and disadvantages of both approaches described above and illustrated in Fig. 2.2, it becomes clear that a viable solution is to use them together to leverage their strengths, as many methods in the literature do [19].

So far, we have abstracted ourselves from the dimensions in which we plan our trajectory. Bringing that into the discussion, we can further realize how broad the concept of path planning is: it has the basis – that is, the classic 2D algorithms that have been effectively integrated into 3D methods over time –, giving rise to numerous other approaches [20, 21].

According to [22], some of the most used methods for global path planning mainly include genetic algorithms, fast random search trees and bee colony algorithms. Also some of the most popular local path planning algorithms are: Artificial Potential Field (APF) methods, algorithms based on fuzzy controllers and scrolling window algorithms, among many others.

Fig.2.3 shows a taxonomy of 3D path planners organized in categories. Sampling based algorithms are among the most common mainly because they avoid the problem of local minima and they provide an adequate compromise between compu-

tational cost and completeness ratio. Therefore, they are used to deal with distinct matters (such as dealing with dynamic obstacles instead of static ones).

APF algorithms are a classic example of sampling based algorithms. They essentially work by dividing the workspace into a discrete grid of cells and applying a set of rules to decide on which path to follow. Let us take for example one of the most well-known and widely used APF algorithms, the Virtual Force Fields (VFF) algorithm [23]. In VFF, after the initial discretization of the workspace, each cell is given a real number between 0 and 1 according to the level of confidence in its occupancy, usually 1 meaning that we have 100% confidence that there is an obstacle in that cell. Lastly, the algorithm uses the position of each cell and the position of the robot to compute virtual forces that act on the robot. There are two types of forces in play: Repulsive forces, in every cell that is above a certain probability of being occupied and pushing the robot away from that specific cell, and there are virtual attractive forces pulling the robot towards the goal point. In the end, the robot manages to navigate through the workspace by following the resulting force vector, that results in the combination of every force affecting the vehicle in each iteration.

Bioinspired and multifusion based algorithms [18] are among the most recent advances in this area. Bioinspired algorithms are based in biological beings and their natural laws and habits, such as the hierarquization of bee hives or ant colonies, Darwin's theory of evolution (applied particularly to evolutionary algorithms) and so forth. Multifusion algorithms are based on two or more simpler algorithms – such as Ant Colony Optimization (ACO) and improved potential fields in [24]) –, that combine into a more complete and robust one.

2.2 3D Navigation in Forest Environments

Within the realm of navigation and path planning techniques, not all of them can be applied in 3D environments and, even inside the feasible group of techniques, they are not all able to cope with forest scenarios efficiently.

Forest environments are, arguably, one of the most challenging terrains to face

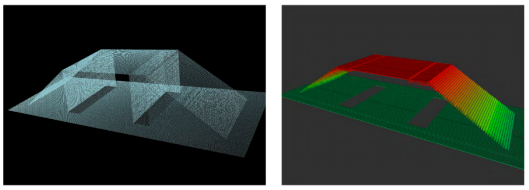
(a) A common pine tree forest.

(b) A cluttered forest.

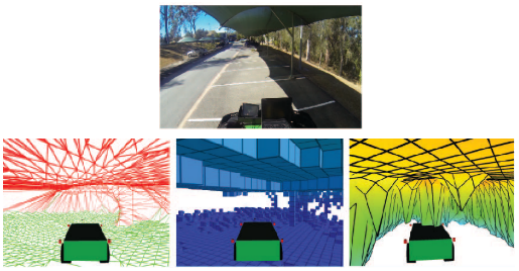
Figure 2.4: Examples of forest scenarios.

when it comes to autonomous robots. On one hand, this type of scenario is **completely unstructured**, the soil can have different shapes and textures, different areas can have totally different characteristics, as shown in Fig. 2.4a, or they can even change throughout the robot's mission in an unpredictable way. As Fig. 2.4b illustrates, trees can be upright, they can be bent, large, small or even fallen. Dense foliage may occlude sensors, and the forest as a whole can hide hard objects of large dimensions that may damage the robot if it considers the terrain as traversable – as is the case with the large boulders in Fig. 2.4b –, among many other examples that could be presented. On the other hand, forest environments are typically cluttered with 3D information, thus generating large amounts of data to be processed, making this process considerably complex and slow, which in turn may lead to its own problems, seeing as navigation algorithms, especially in forests, must be fast and efficient when generating outputs. In some cases, robots may encounter danger before the map is updated with that information. Other factors that make this environment harsh when it comes to autonomous navigation include: the bad signal reception from satellite positioning devices, hindering localization of the robot; or the steep angles of the terrain, limiting the robot's maneuverability.

Grouping the challenges of navigating in this kind of environment, we are interested in three main fronts related to this application:



(a) 2.5D-NDT representation of a bridge. Adapted from [25].



(b) OVPC Mesh compared with OctoMap and Elevation Map, respectively. Adapted from [26].

Figure 2.5: Examples of different representation methods.

- **Environment Representation** - We must choose/develop an efficient technique to represent the foreseeable space with minimal resolution and fidelity loss while still maintaining performance computation wise.
- **Traversability Analysis** - We must choose/develop an appropriate method to analyze the generated representation in order to isolate traversable from non traversable areas.
- **Path Planning** - We must design/choose an appropriate method that enables the robot to navigate through traversable space until it reaches the desired goal point.

Environment Representation Techniques

Elevation Map [27] - This method incorporates the drift and uncertainties of the robot's state estimation, as well as a noise model of the distance-measuring sensor in order to create a probabilistic terrain estimate as a grid-based elevation map, with upper and lower confidence bounds.

2.5D-NDT [25] - This method is proposed as a 2.5D Normal Distribution Transform (2.5D-NDT) map. Its main benefits are that it only stores information about points that are believed to be traversable by ground robots, as clearly shown if Fig.

2.5a, the resulting map is efficiently organized according to a *proprietary* two-index system designed by the authors, speeding up the computations.

2D Costmap [28] - This is among the pioneer methods used to efficiently represent the surrounding environment in such a way that a mobile robot can navigate accordingly. The space is essentially discretized into a 2D grid of a chosen resolution, then the information in the Z-axis (if existent) is projected into the 2D plane and each cell is given a value according to the existing belief about its occupancy. Then, obstacles are typically inflated in order to guarantee a safety distance for the robot and are lastly fed to the path planner to generate motion commands around these obstacles.

OVPC Mesh [26] - On Visible Point Clouds (OVPC) Mesh is a method that conservatively represents the visible spatial surroundings of the robot as a watertight 3D mesh that aims at providing reliable 3D information for path planning while fulfilling real-time constraints. This approach provides a satisfactory trade-off between representation accuracy and computational efficiency, resulting in a technique which is faster than both octomap and elevation map, while still correctly categorizing harder obstacles such as overhangs and thin poles, as observed in Fig. 2.5b.

Navigation Mesh [29] - The Navigation Mesh method uses 3D point clouds to reconstruct a triangle mesh of the environment in real time that also has local connectivity information. This resulting mesh is then analyzed for roughness and traversability and fed to the path planner.

OctoMap [30] - This method generates volumetric 3D environment models based on octrees, which are a data structure optimized for 3D models, thus achieving a considerable reduction in computational complexity when compared to other volumetric representations. The 3D sensed environment is discretized in 3D cells of a given size, making this method excel in representation fidelity, but incurring in heavy computations and large times for extensive and complex environments when compared 2.5D methods, such as elevation maps.

Mechanical Effort Based Map [9] - This representation technique is among the most recent to be proposed, it distinguishes traversable from non-traversable areas, computes a mechanical effort measurement of the robot in the given terrain according to its 3D gradient, and fuses the data in a 2D costmap.

The area within a given distance around the robot is analyzed (ideally at run time), the gradient of the terrain is calculated and the concept of mechanical effort is introduced based on that. Then, the calculated mechanical effort is used to populate a 2D cost map which is later used for navigation.

Traversability Analysis Methods

Semantic Segmentation [31] - This is one of the most recent techniques of categorizing terrain as traversable. Neural networks are trained to take as input an image, categorize every pixel into a class and also return a 2D image where every pixel has an assigned class. This output is generated in the camera's frame of reference, which means that it needs to be subject to a coordinate frame transformation before it can be used as input for a navigation system.

3D OctoMap based Grid Map [32] - The perceived world is first discretized using an octomap, and that octomap is then divided in horizontal layers of a set height defined by the user. Every layer is then analyzed considering inter-layer dependency in order to identify traversable portions of terrain (for instance, levelled areas or slopes) and non-traversable portions of terrain (for instance, cliffs or hills that the robot cannot safely climb).

2.5D-NDT Traversability Analysis [25] - The area previously divided into patches by the corresponding spatial representation technique is then analyzed in order to find which of the patches are traversable. This analysis is done by considering a $2r * 2r * 2r$ sphere as the robot spatial footprint, checking if there are patches inside that sphere and, in case there are, comparing the height of neighbour patches relative to each other and to the robot, to check if that patch of terrain is suitable for navigation.

Neural Network Depth Maps [33] - Similarly to semantic segmentation, this approach also uses neural networks. However, the output of the networks are directly navigation commands, instead of semantic classes that need to be post-processed to generate the desired navigation commands.

Path Planning Techniques

Virtual Force Field [23] - A method that discretizes the workspace in cells, and assigns a force vector to each cell (repulsive force if the cell has an obstacle, attractive force if the cell has the objective point, zero if the cell is empty) and then navigates according to the resulting force vector.

Vector Field Histogram [34] - A similar method to VFF, but discretizes the workspace according to an occupancy histogram instead, pursuing the path that has the biggest unoccupied area.

Neural Networks [35] - In general, this type of algorithms require a very complex training stage with, for instance, input labelled data (images for segmentation, navigation commands, etc.) in which they try to approximate the behaviors and patterns present in the training data, so that in the presence of new data, it can generate an appropriate output.

Fuzzy Controllers [36] - This type of algorithm is built using the knowledge and experience of humans about the process to be controlled. Based on this knowledge, a set of rules is created, and the inputs of the system are fed into a fuzzifier, responsible for transforming continuous variables in fuzzy variables, which then suffer the effects of the inference engine, being lastly reconverted into continuous variables by the defuzzifier. This process allows us to control systems from which the mathematical model is hard or impossible to extract.

Sampling Based Algorithms [18] - In algorithms like RRT (Rapidly-exploring Random Tree), the configuration space is rapidly explored by creating a new random

node in each iteration of the method, and testing if it is possible to traverse into that new node while remaining inside the configuration space. If this test is sucessful, the newly created random node will be saved and become attached to the previous one. This process is repeated until a path reaching from the starting to the goal pose is obtained.

Node Based Algorithms [18] - In algorithms like A* and Dijkstra's, the environment is sampled into a graph composed by nodes and arcs and the method iterates by advancing to the next connected node of minimal cost, eventually reaching the goal pose. The generated path is thus guaranteed to be the one of minimum cost, according to the metric used.

2.3 Potential Impact of This Work

As stated by [37] and on the authority of a study from the European Commission, road transports account for 72% of the overall EU greenhouse gas emissions, thus reassuring the importance of developing new and more efficient fuel-related technologies [38, 39, 40]. Among the research being undertaken on the topic, many articles have shown that there is a strong correlation between the fuel consumption of a vehicle in a certain terrain with the gradient and the roughness of that terrain – even when considering more common types of vehicles like passenger vehicles in asphalt roads [39, 41, 42].

According to [42], fuel costs make up to 35% of the forestry sector expenses in Sweden. Furthermore, terrain gradient and road grade account for more than 77% of the variation in fuel consumption of forestry vehicles, being terrain gradient the main factor [42, 41]. Also, from the analysis of a dataset collected in a real world scenario with a heavy duty logging truck, there is a clear relationship between the characteristics of the terrain and the fuel consumption of the vehicle, as shown in Fig. 2.6. These considerations help to sustain the relevance of the proposed algorithm which strives to choose the path of less effort, thus reducing the energy requirements for the locomotion of the robot.

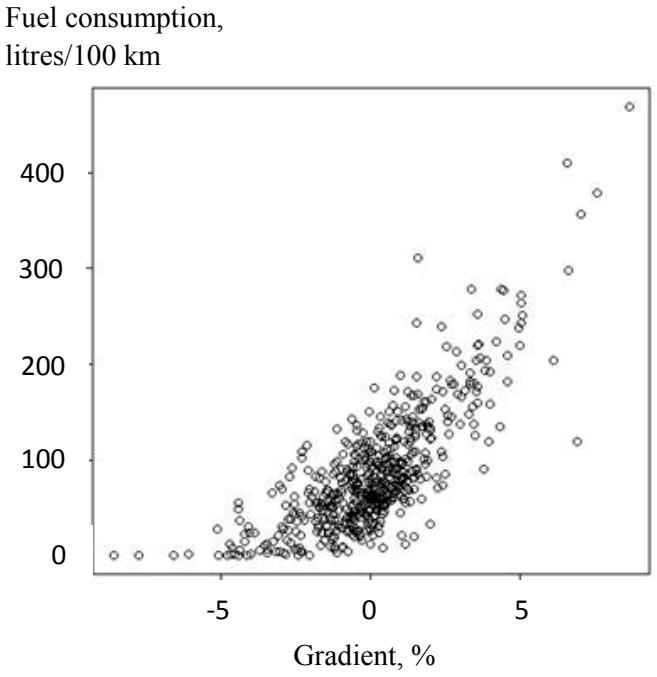


Figure 2.6: Relationship between terrain gradient and fuel consumption on a heavy duty forestry vehicle. Image taken from [42].

On a more speculative note, using the data collected and processed by [42], we can assume with some degree of certainty that the correlation between the gradient (G), International Roughness Index (IRI) and fuel consumption shown in the following equation[1]:

$$Fuel = 46.19 + 22.33 * G + 1.47 * G^2 + 7.70 * IRI, \tag{2.1}$$

is approximately proportional, thus inferring that the results of the same tests with our robot would simply be scaled down, while maintaining the distribution of the values relatively constant, according to

$$Fuel = K * (46.19 + 22.33 * G + 1.47 * G^2 + 7.70 * IRI), \tag{2.2}$$

where K is the unknown scale factor between the two distributions.

[1]Equation taken from [42].

2.4 Research Gaps and Contributions

In this section, we discuss the existing techniques for representation, traversability analysis and path planning and study how they compare with one another, as well as with our expectations of what an all-emcompassing method should be, aiming at finding the most promising one.

In Table 2.1 we compare some of the existing methods used for spatial representation, i.e. the way how algorithms sample the real (or continuous environment) into some sort of discrete representation. In our comparison, we found some metrics to be the most appropriate ones to describe this group of algorithms based on the available information, namely:

- Resolution Loss: The amount of detail a method can capture and represent is a fundamental specification of any environment representation algorithm, and so is the amount of detail it eliminates. This metric represents how much detail is lost when applying the method.

- Dimensions: Although the number of represented dimensions can be higher than three (for instance, 4D - three spatial dimensions and one temporal dimension), we chose an interval between two and three dimensions for this comparison.

- Computational Cost: This is an essential metric for any real-time or close to real-time method, given that it directly represents the complexity of all the calculations involved and the relation between the complexity of the environment and the duration of the representation process.

- Test Environment(s): Whether a proposed method is tested under real circumstances or not is an important information, given that even the most realistic of the simulators cannot perfectly emulate reality. On the other hand, testing software on a realistic simulator is of major importance, given that we can replicate extremely challenging conditions, and collect valuable data without worrying about damaging expensive equipment.

- Tested in Forest Environments: Bearing in mind the considerations explained earlier about the specificity and complexity of forest environments, this metric specifically indicates whether a given method was tested in forest environments.

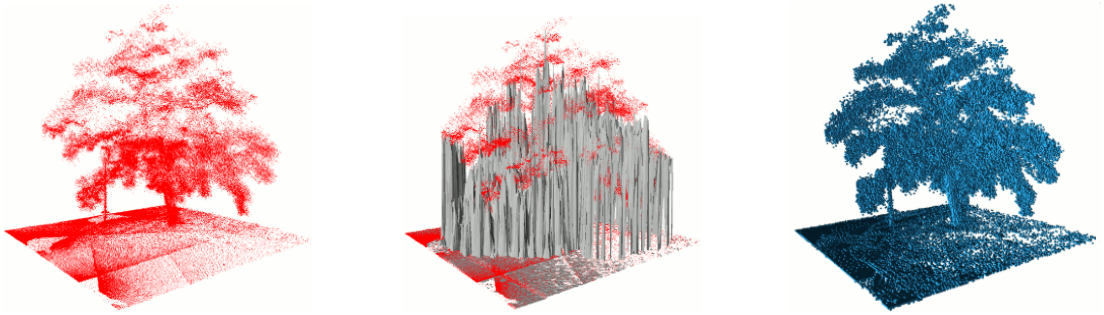
As we can see, the majority of the techniques shown in Table 2.1 use at least 2.5D, with only the 2D costmaps and the mechanical effort approach using less than that.

Generally, the more dimensions a method uses, the higher the fidelity of the representation is. However, this has a big impact on the computational cost.

For instance, looking at the methods from the first two rows of Table 2.1, we can see that both represent the world in 2.5D, however one is considered to have a higher resolution loss than the other. This is because of the way each method processes the received 3D information. As explained earlier, *Elevation Map* creates a triangle-based mesh where the higher point in the Z-axis in each cell defines the height of that particular triangle, making the representation of some structures like bridges impossible (this is visible in Fig. 2.7b, where nearly all the terrain under the tree is classified as non traversable, when in fact that is not true). On the other hand, the *2.5D-NDT* method groups the points from a point cloud into discrete patches and saves several horizontal patches, which makes it possible to have two parallel patches, one on top of the other, being able to represent a higher amount of structures (such as bridges).

Similar conclusions can be inferred about the two 2D methods presented in Table 2.1. Although they both are 2D, *mechanical effort based maps* capture way more information about the environment than simple *2D costmaps* do. Once again, we can see that this reflects strongly on the computational cost. Therefore, it becomes apparent that a method cannot be the best at every category; it must, however, consist on a proper combination of categories, prioritizing the most relevant aspects considered.

Having this in mind, as well as the entire analysis available on Table 2.1, we believe that *2.5D-NDT*, *OVPC Mesh* and *Mechanical Effort Based Map* are the methods that show the higher potential for our particular application.



(a) Point cloud. (b) Elevation map. (c) OctoMap representation.

Figure 2.7: Examples of two different representation methods and the original point cloud. Image adapted from [30].

Table 2.1: Comparison of methods for spatial representation.

Name	**Brief Description**	**Resolution Loss**	**2D/2.5D/3D**	**Computational Cost**	**Test Environment(s)**	**Tested in Forest Environments**
Elevation Map [27]	Triangle-based 2.5D elevation costmap	Medium	2.5D	Medium to High	Simulation and in real robots, both indoor and outdoor	Yes
2.5D-NDT [25]	2.5D-Normal Distribution Transforms Map, which divides the world in so called patches.	Low	2.5D	Low to Medium	Simulated and real datasets	No
2D Costmap [43]	Map that discretizes the world in cells and fills each cell with 2D information about the existence of obstacles in the projection of the world that results in that 2D cell.	High to Very High	2D	Low	Extensively tested in all kinds of environments	Yes
OVPC Mesh [26]	On Visible Point Clouds Mesh is a method that conservatively represents the visible spatial surroundings of the robot as a watertight 3D mesh.	Low	3D	Low to Medium	Simulation and real robot	Yes
Navigation Mesh [29]	Navigation Mesh method uses 3D point clouds to reconstruct a triangle mesh of the environment in real time that also has local connectivity information.	Low	3D	Medium	Real robot	No
OctoMap [30]	OctoMap generates volumetric 3D environment models based on octrees.	Almost none	3D	High to Severe	Extensively tested in all kinds of environments	Yes
Mechanical Effort Based Map [9]	Distinguishes traversable from non-traversable areas, computes the mechanical effort of the robot in the given terrain and fuses the data in a 2D costmap.	Medium	2D	High to Very High	Simulation	No

Table 2.2: Comparison of methods for traversability analysis.

Name	**Brief Description**	**2D/3D**	**Computational Cost**	**Local/Global Analysis**	**Test Environment(s)**	**Tested in Forest Environments**
Semantic Segmentation [31]	Performs a pixel-wise analysis and labels each pixel as part of a given class (e.g. road, tree, bush,...)	3D	Medium to High	Local	Real and Simulated	Yes
3D OctoMap based Grid Map [32]	Divides an octomap that represents the perceived environment into layers and analyzes the layers relative to each other to infer traversability features.	3D	High to Very High	Global	Real and Simulated	Yes
Elevation Map [27]	Adjacent cells are analyzed with respect to some metrics, for instance, the height difference between them, and a traversability map is generated.	2.5D	Medium	Local	Real and Simulated	Yes
2.5D-NDT Traversability Analysis [25]	The patches previously identified by the respective spatial representation method are analyzed to extrapolate a traversability map, performing several routines.	2.5D	Low to Medium	Global	Real and Simulated	No
Neural Network Depth Maps [33]	Although this approach also uses neural nets, unlike semantic segmentation, the output of the nets are directly navigation commands.	3D	Medium to High	Local	Real and Simulated	No
Local Roughness Estimation [29]	Infers the continuity of the surface its respective roughness value, choosing then the path with less roughness overall.	3D	Medium to High	Local	Real and Simulated	Yes

Table 2.3: Comparison of methods for navigation and path planning.

Name	**Brief Description**	**2D/3D**	**Computational Cost**	**Local/Global Planning**	**Test Method(s)**	**Tested in Forest Environments**
APF [18]	Artificial Potential Fields - Methods like VFF and VFH that discretize the workspace into 2D cells, apply virtual forces to each cell and navigate following the resultant force vector.	2D	Low	Local	Simulation and in real robots	Yes
Mechanical Effort [9]	Creates a local plan that follow the path that requires less mechanical effort from the robot, according to the previously generated map from the same method.	3D	Very High	Local	Simulation	No
Neural Networks [35]	This kind of algorithms take, for instance, labelled images as the input, and return the semantic classes each pixel belongs to.	3D	High on training stage Lower on running stage	Depends on the implementation	Simulation and in real robots	Yes
Fuzzy Controllers [36]	This kind of algorithms uses a rule basis to calculate the output. Such rule base is built with linguistic variables and according to prior knowledge/experience of humans.	Both	Medium	Both	Simulations and in real robots	No
Sampling Based Algorithms [18]	Algorithms like RRT (Rapidly-exploring Random Tree) where the configuration space is rapidly explored in order to find a path leading from the start to the end goal.	Both	Medium to High	Global	Simulation	No
Node Based Algorithms [18]	Algorithms like A* and Dijkstra's, which deal with graphs composed by nodes and arcs and calculate the cost of exploring the nodes, eventually reaching the optimal path.	Both	Medium to High	Global	Simulation and in real robots	No

The analysis of Table 2.2 is similar in nature to the one just performed and the general conclusions still apply. It is important to note that the "**Resolution Loss**" metric was swapped with "**Local/Global Analysis**" as we consider the latter to be more appropriate when it comes to traversability analysis. In this context, we consider that a method performs a local analysis of the environment when it has to plan using only the information given from the onboard sensors about the surrounding perceived environment at each particular instance, and a global analysis when the method has information about the environment that is, at that point in time, outside of the sensory horizon of the robot, either by having an *a priori* map of the environment or by recording previous sensor reading in some sort of representation.

Although implementations based on neural networks show promising results in some cases, we consider it too resource intensive for this particular work, considering all the training labeled data it would involve in order to have a chance of succeeding. From all the methods described in Table 2.2, we believe *2.5D-NDT Traversability Analysis* to be the most promising one, considering its ability to navigate through challenging environments while only being 2.5D, thus making the computational cost of the process lower when compared to other methods.

In Table 2.3 we present some of the existing approaches for navigation and path planning. It is noteworthy that these are just a few of the most common ones, many others based in different concepts could be mentioned, such as evolutionary algorithms, bio-inspired algorithms, e.g. bee colony and ant colony algorithms and even some based in bacterial foraging theory [44].

Although the *mechanical effort* method is considered to have a very high computational cost, we believe the concept is extremely interesting and, with proper modifications and expansion to a global analysis, it can become a much more efficient method.

Similarly to the resulting conclusions from table 2.2, *neural networks* are considered too expensive data wise to be accounted as a valid candidate for this work.

Both sampling and node based algorithms are valid candidates, yet they tend to generate results slowly in extensive and cluttered environments, like the ones we

intend to deal with.

When it comes to *fuzzy controllers*, we consider them as possible candidates in simpler environments. However, in this work we are dealing with highly complex forest environments and, as such, we believe that the performance offered by this technique is not adequate as the number of rules to correctly navigate would be too high.

Based on the analysis performed, we believe that *mechanical effort* based navigation, *sampling* and *node based algorithms* are the more promising ones, as they have the most fitting ratio of the used metrics to our needs.

Taking into account the tables presented as well as the analysis performed, we consider the following points as existing research gaps in the state of the art at this moment:

- Mechanical effort based traversability analysis shows, not only what is and is not traversable, but also a very interesting way of deciding which path is the most efficient one, especially with regards to fuel economy. However, this method has been developed for local planning, which makes it less computationally efficient and limits its success.

- 2.5D-NDT method presents a very promising approach for the specific task of spatial representation for vehicle navigation, considering the ratio of computational cost/representation detail it presents. This was not, however, tested in a real environment nor in a forest one.

- There is, currently, lack of works with proven effectiveness in forest environments for spatial representation, traversability analysis and navigation.

- The current literature in forestry robotics lacks fuel related research, such as fuel consumption models and fuel economy techniques.

Having the previous research gaps in mind, we consider the following points to be relevant candidate contributions of this work:

- A thorough state of the art review on the subject of navigation, path planning and traversability analysis of UGVs, especially in unstructured environments such as forests;
- Expand the mechanical effort method [9] to perform a global analysis, with the intent to improve its overall performance and thus generating a robust method for traversability analysis and navigation.
- Extension of the mechanical effort algorithm with a mechanism to deal both separately and concurrently with a lethal obstacle layer, which we called *evident obstacle layer* and a *terrain roughness layer* that is later fused into a single costmap.
- Optimize the implementation of the algorithm so that it can execute in real time.
- Benchmarking with other existing techniques to prove the validity of the proposed method.

3 System Architecture

3.1 Overview

This Chapter describes the developed architecture in detail. Fig. 3.1 presents its overview, with the intended dataflow through the main areas of focus considered. Our system consists of:

1. A pre-processing module responsible for downsampling the input data;
2. A traversability analysis block responsible for generating traversability maps;
3. A navigation technique that uses its contents to generate and execute paths.

The system uses a so-called pointcloud map – a pointcloud representation of the map generated with any pre-existing SLAM technique – of the environment as its input, that starts by undergoing multiple steps of decimation and pre processing (Section 3.4). The output of these inner stages (Section 3.5) are then used as input to the two main stages of our system: the *evident obstacle detection* (Section 3.6) and the *terrain roughness estimation* (Section 3.7), where in each one a costmap containing the respective information is generated. The two costmaps that result from these stages then serve as input for the data fusion stage (Section 3.8) which is responsible for generating a single costmap that contains all the relevant information. This costmap is finally fed into the navigation stack that uses it to compute the global path and drive the robot accordingly (Section 3.2). Thus the robot acts on the world, changing its configuration, which generates new sensory data, and thus closing the loop.

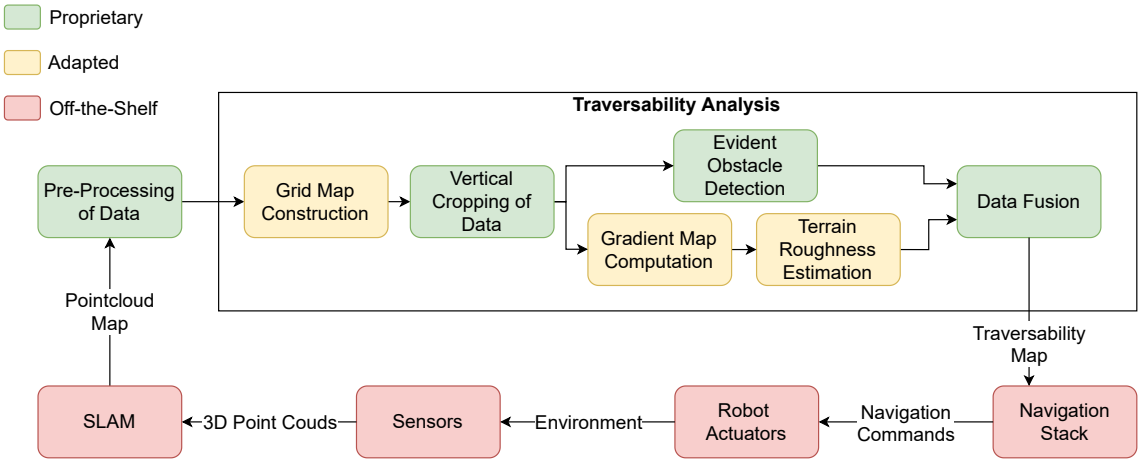


Figure 3.1: System architecture overview for proposed solution. Green boxes represent modules implementing completely novel algorithms/processes, yellow boxes represent modules implementing existing solutions modified for the purpose of this work, and red boxes represent off-the-shelf modules by other authors used "as is".

Fig. 3.1 represents the novel architecture proposed by this work. It is composed of several modules, from which some have been created from scratch, while others have been adapted or simply used.

In virtue of having built the entire system using Robot Operating System (ROS) nodes, it runs concurrently, meaning that a new map may start being processed while the final output of the previous one is not yet generated. There is, however, one particular stage of the pipeline, comprised by the modules described in Sections 3.6 and 3.7, that is able to execute in parallel to the same input map.

3.2 ROS - Robot Operating System

ROS [45] is a framework designed to help in the development of software for mobile robots. It has a conceptually simple yet very powerful workflow, shown in Fig. 3.2, with its main features being:

- Ability to abstract from the hardware side of the problem and focus on the software side;
- Native synchronous and asynchronous communication protocols;

- Ability to generate self-contained units of software (packages) that can be interchangeable between projects with minor or no modifications required.

ROS has four main components: Packages, Nodes, Communication Protocols (services and topics) and Messages.

Packages in ROS are essentially a directory which contains the necessary software to implement a given feature, which can, for instance, translate into empowering our robot with an additional feature (such as a PID controller or some sensor's driver), ideally with little to no adjustments. A package might contain ROS nodes, a library, a dataset, configuration files, a third-party piece of software or anything else that constitutes a logical module when put together.[1]

When packages are compiled, generally they define independent executables called **nodes** that are created during compilation, which are processes that can communicate with each other to generate the required data flow for the correct behavior of the implemented system. Nodes can then **publish** or **subscribe** to a **topic**, which enables them to access required information that other nodes are sharing, or do themselves the sharing of information.

Topics are a one way communication system bwtween nodes: although they can either have a node publish information on them or have a node subscribe to it to get the information it contains, the flow of information on the topic is **always** from the publisher to the subscriber.

Both topics and **services** provide communication between nodes. However, while topics do it in an **asynchronous** way (nodes publish information on topics and other topics may or may not subscribe to it), services implement a **synchronous** communication protocol, requiring both a **request** and a **response** to be transmitted between the communicating nodes in order to succeed.

All the previous communication is done with the help of **messages**, which are pre-specified data containers composed by typed field of various complexities, from a single integer to nested arrays of primitive types.

Fig. 3.2 illustrates a possible implementation of a system running with ROS.

[1] http://wiki.ros.org/Packages

[2] http://wiki.ros.org/Master

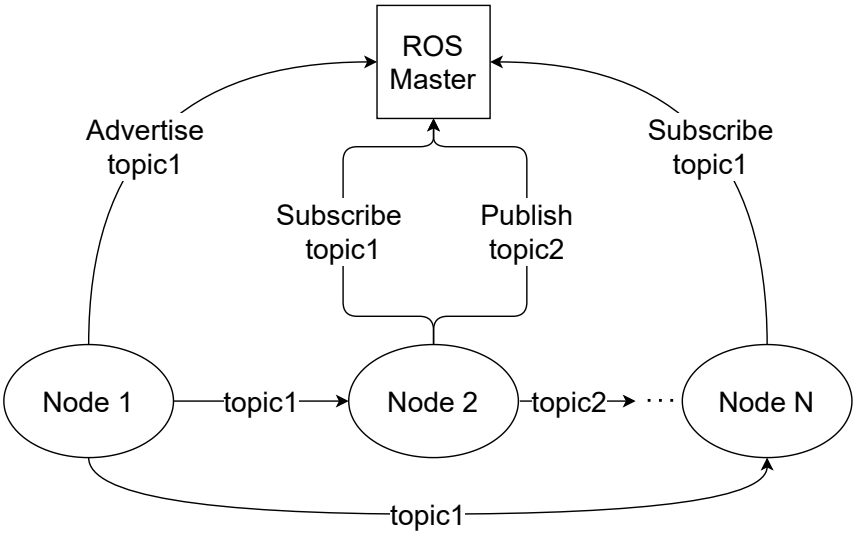


Figure 3.2: ROS communication paradigm (inspired in the ROS official page[2]).

It has a few nodes that advertise (publish) topics, i.e. they notify the master node about wanting to publish a certain type of data into a certain topic and then, the master node enables the data to be published (with no data transfer yet). When some other node(s) subscribe to the same topic to acquire relevant information, the data is then transferred, the subscriber node receives it and a peer-to-peer connection is created between those nodes. Such connection is later eliminated if it is explicitly closed by one of the nodes.

Given that ROS provides a simple yet powerful way of programming a complex system – such as a mobile robot – described above, while greatly facilitating the integration with simulation tools and having an extremely active community, we use it in this work for all the implementations and experiments described.

Navigation Stack [43] is a collection of ROS packages that endow a generic mobile robot with the capability of navigating in 2D environments. Even though it is not a one-fits-all solution, it can also be tuned to work in 3D environments and produce satisfactory results. This is what we did in order to process the final costmap that results from our system and convert it into movement commands.

Instead of using observation sources to populate the costmaps, the navigation algorithm is fed with the final map produced by our algorithm through a ROS plugin called *costmap_2d::StaticLayer* into its global costmap, which is then used

to generate the path. Considering that the scope of this work is global planning, the local costmap is not populated nor updated, it simply stays empty. However, this does not mean that the local planner is not used, as it actually guides the robot through the terrain, closely following as much as possible the global path generated earlier.

3.3 Grid Map Construction

This module is responsible for creating the 2D grid map that serves as input to other modules. It takes as input the pointcloud map that has been processed in the first step of data cropping and creates grid of configurable size where each point in the map is projected and assigned to a particular cell according to its location in space, as illustrated by Fig. 3.3.

The "Grid Map Construction" block referenced in the architecture (see Fig. 3.1) was adapted from [46]. The original module was analyzed with a profiling tool (`cProfile`), which timed all components of the system during its execution. After that, we have used `pStats` to illustrate how much processing time is dedicated to each component of the system. This analysis revealed a major computational peak in the algorithm - a very resource-intensive loop that we managed to significantly improve by using python iterators such as `enumerate` and `zip` instead of the typical three dimensional nested `for` loop that was implemented. This drastically reduced the complexity of the algorithm, addressing and solving the first of [46]'s future work proposals. In Fig. 3.4 we present a plot that shows the scale at which the algorithm's complexity was decreased. Although this figure only compared the map generation time with the size of the pointclouds, it is also worthy to mention that now the execution time of the algorithm also scales much better with the size and the resolution of the map, transforming what was a rapidly accelerating exponential complexity into an almost linear one.

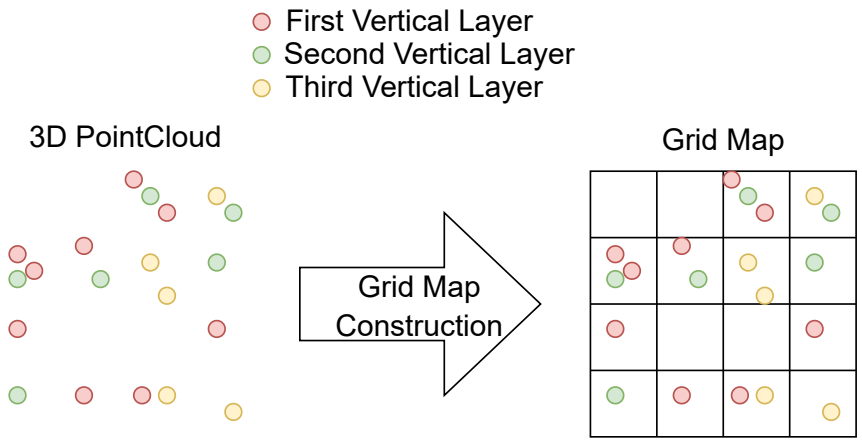


Figure 3.3: Illustration of the grid map construction stage (top view of the point-cloud).

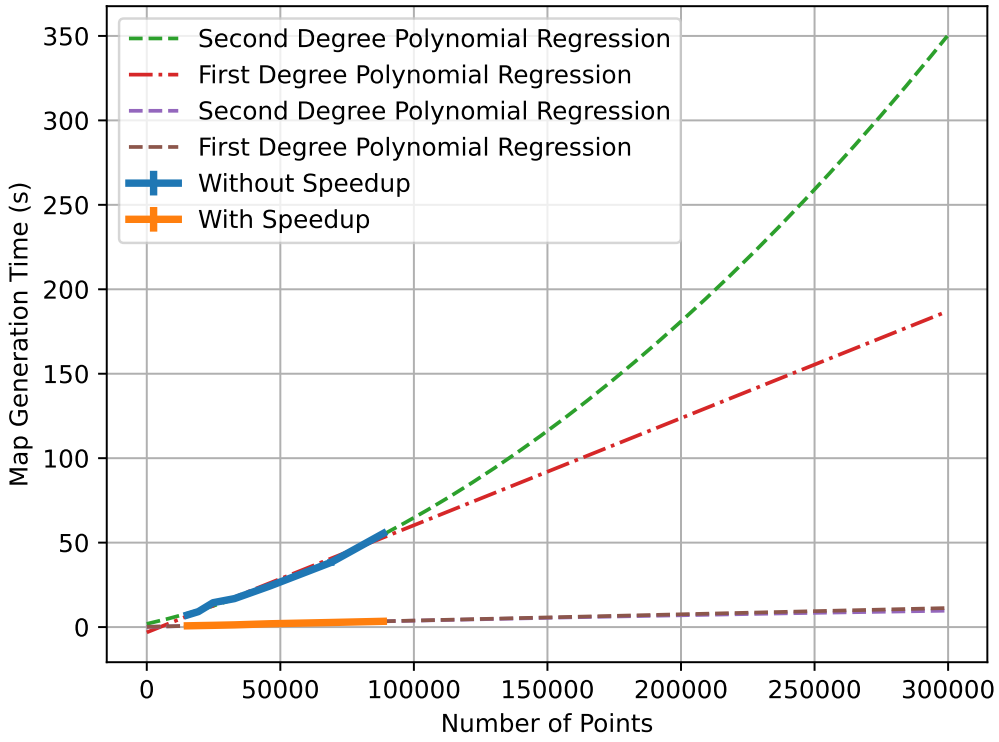


Figure 3.4: Comparison between the map generation speed of both systems.

3.4 Pre-Processing

This stage of the pipeline is fed a pointcloud map of the complete environment received from any SLAM technique, which is not the scope of this work, and filters the received map in order to exclude the points that are outside the workspace of the robot. There are four main steps within this stage:

Data Decimation - The raw pointcloud is decimated using a 3D voxel grid with a configurable downsample factor, which aproximates all the points contained within each voxel with their centroid, i.e. the spatial average of the points in the cloud confined by each voxel. This significantly reduces the number of points in a pointcloud (up to 6.5x lower), depending on the point density that our system requires. This functionality was implemented using the Point Cloud Library (PCL) `VoxelGrid` filter. PCL is a standalone, large scale, open project for 2D/3D image and point cloud processing[3].

Change of Frame of Reference - This step is necessary to make sure that the cropping described in the following points is centered in the robot's frame of reference and not in the world reference frame. This step is of major importance, considering that we intend to have a scrolling window of information centered in the reference frame of the robot and, if it was centered in the fixed world frame, at some point the robot would exit the cropbox of known information. To implement the algorithm described above, we have created a C++ ROS node that takes advantage of the efficient PCL algorithms (available in the `pcl_ros` package) and modifies the frame of reference of the entire input pointcloud, from the world frame of reference to the robot's.

Horizontal Data Cropping - Receiving as input the decimated pointcloud in the robot's frame of reference, this module filters all the points that are outside of a three-dimensional box of a configurable size and no limit along the Z-axis, considering that these points do not have a significant impact in the behavior of the

[3] `https://pointclouds.org/`

robot. The importance of using cropbox with no Z limit in this stage is depicted in Fig. 3.5a, where we can observe that, by cropping along the Z-axis up to a certain height, it is only reasonable when the terrain is levelled. As soon as we introduce hills on the terrain, this technique is no longer guaranteed to work, given that we will be cropping the entire pointcloud at the same level with no regards to the ground level for each cell. That may result in the elimination of unwanted information, such as higher ground trees and/or too little of the lower ground ones, leading to unreliable path planning.

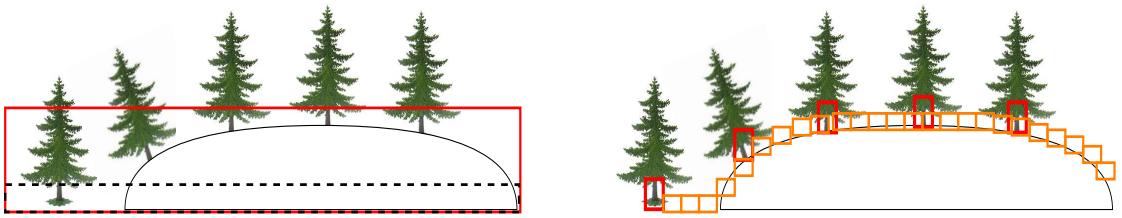
Vertical Data Cropping - This step deals explicitly with the vertical axis of the pointcloud. Considering that at this stage we already have a discrete map of $N \times N$ cells, we perform a cell-wise analysis where we check which is the lowest point in each cell, and then only consider the points that stand up to a given amount above it. This heuristic is demonstrated in Fig. 3.5b and can be summarized by

$$\text{point is } \begin{cases} \text{not discarded } \textbf{if } z\_current\_point - min(z\_all\_points) < threshold \\ \text{discarded } \textbf{otherwise} \end{cases},$$

where the value for the specified threshold is adjusted empirically and varies depending whether we are dealing with the evident obstacle layer (Section 3.6) or the terrain roughness estimation layer (Section 3.7).

In Fig. 3.6, we show the effect of the chosen heuristic's application. Fig. 3.6b shows the costmap that results from not applying the vertical cropping heuristic, while Fig. 3.6c shows the resulting costmap from the full pre-processing stage, including the chosen vertical cropping heuristic.

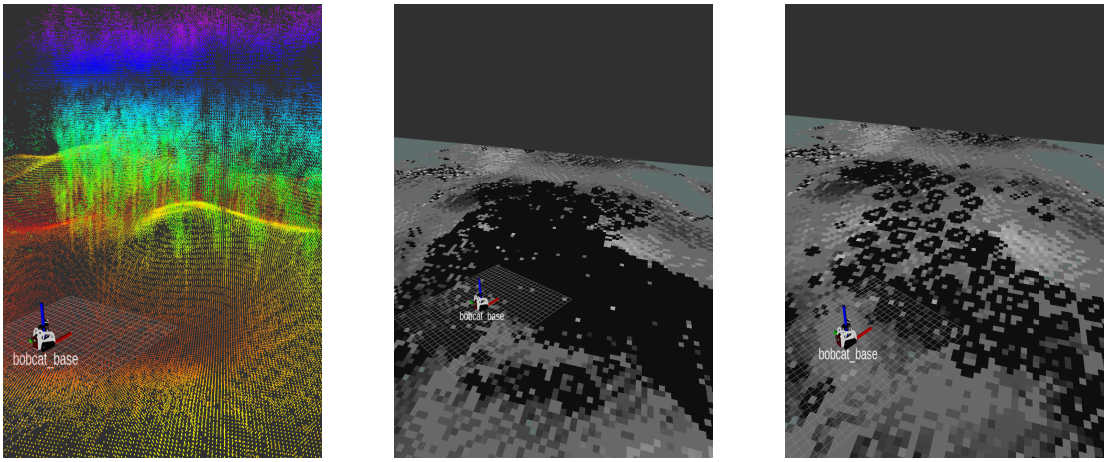
The main goal of this stage is to reduce the number of points to be further analyzed and consequently reduce the complexity and computational cost of the entire process.



(a) First vertical heuristic: cropbox.

(b) Second vertical heuristic: cell by cell analysis.

Figure 3.5: Comparison between the two cropping heuristics. The heuristic illustrated in (b) was ultimately integrated in the system.



(a) Pointcloud to be analyzed.

(b) Costmap before applying the vertical cropping heuristic.

(c) Costmap after applying the chosen vertical cropping heuristic.

Figure 3.6: Costmap representation (XY plane) before and after applying the cropping heuristic.

3.5 Gradient Map Computation

This is the first step regarding the actual traversability analysis process, that uses as input the data generated in the previous pre-processing stages and in which the terrain is analyzed through the gradient at each point. To accomplish this goal, we start by making sure that every cell of the grid only contains one point, using the *median* of the sample (the set of points in that particular cell) when that is not the case initially. After this is done, we apply the gradient to the resulting pointcloud, thus generating a gradient map that can be used to estimate the overall roughness of the terrain.

The gradient of a function or, as in this case, the approximation of the gradient of a discrete distribution of points provides us with the information of the direction of greatest variation. As we stated before in Section 3.5, there is a strong correlation between the gradient of a terrain and the energy consumption that it requires to be traversed. As such, using terrain gradient information in this work is of particular relevance, considering that our main goal is to minimize the traversing effort and the fuel consumption.

The general two dimensional gradient vector is defined according to Eq. 3.1:

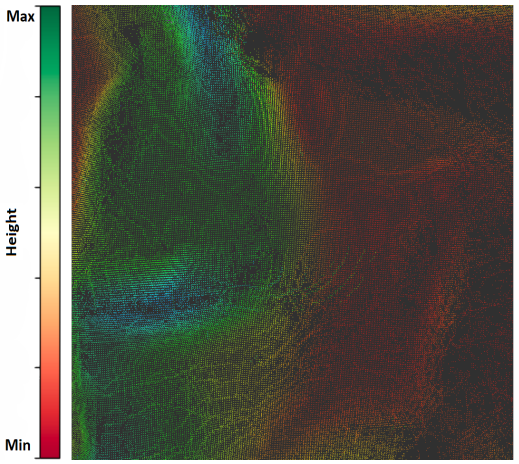
$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}, \tag{3.1}$$

where f represents the mathematical function of which we would like to know the gradient. However, considering that it is not possible to find a function that accurately describes the entire point distribution, the gradient we use is an approximation that is computed using second order accurate central differences in the interior points through:

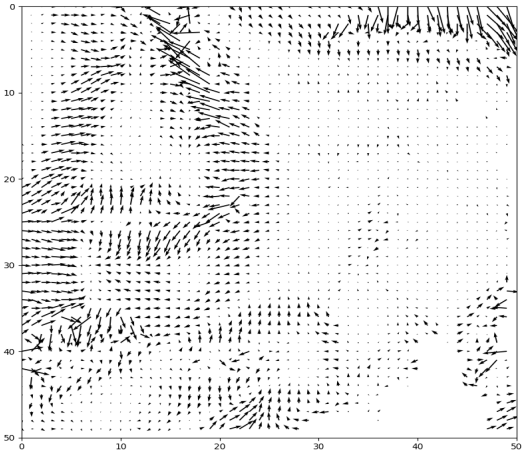
$$\frac{\partial f}{\partial \lambda} = \frac{f(\lambda + h) - f(\lambda - h)}{2h}, \tag{3.2}$$

and first order accurate one-side differences at the boundaries:

$$\frac{\partial f}{\partial \lambda} = \frac{f(\lambda + h) - f(\lambda)}{h}, \tag{3.3}$$



(a) Pointcloud of hilly terrain, where colder colors represent the higher terrain, and vice versa.



(b) Resulting gradient map.

Figure 3.7: Representation of the gradient analysis stage output.

for the left boundaries and

$$\frac{\partial f}{\partial \lambda} = \frac{f(\lambda) - f(\lambda - h)}{h} \tag{3.4}$$

for the right boundaries, for each of the dimensions of the gradient.

The resulting gradient values are then clipped to a configurable threshold, i.e.:

$$\nabla f(x_m, y_n) = \begin{cases} \nabla f(x_m, y_n), & |\nabla f(x_m, y_n)| < \nabla_{th} \\ \nabla_{th}, & \textbf{otherwise} \end{cases}, \tag{3.5}$$

where $\nabla f(x_m, y_m)$ represents the gradient in a given cell and ∇_{th} represents the clipping threshold. This allows us to automatically exclude both hills and depressions that exceed this configurable inclination limit.

This block was adapted from [46] and had to be debugged, considering that there were significant gradient map variations with no apparent cause. The error that was causing this abnormal behavior was an ill-formed flux control loop, where the unitary gradient vector was being calculated according to $unit\_gradient =$

$gradient/norm(gradient)$. That equation is naturally suitable in every scenario except for a planar surface. In such surface, we would be dividing a value by zero and thus generating an indefinite value and consequently erroneous values in the corresponding cells of the final costmap. This problem has been solved and the results improved significantly.

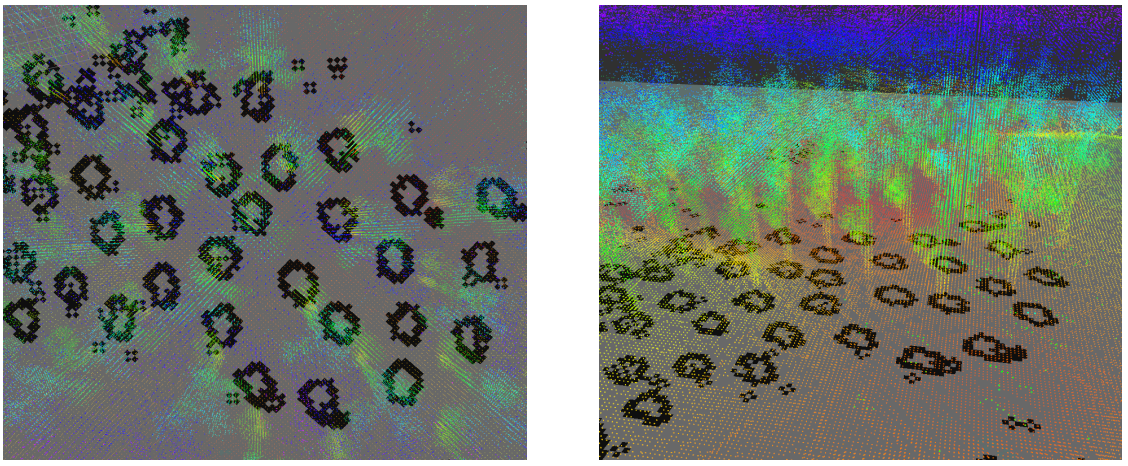
3.6 Evident Obstacle Detection

In this stage, we deal with the grid map that has been built using the data contained in the pointcloud that has been already pre-processed. This data is organized as a tri-dimensional matrix of configurable size that holds the height of all the points that are contained within each cell. We calculate the mean, variance and range of the array of points that form each cell of this matrix and apply the following heuristic that marks a cell as an evident obstacle:

$$obstacle = \begin{cases} \text{true}, & \begin{cases} \mu_{(x,y)} > \mu_{th} \\ \sigma^2_{(x,y)} > \sigma^2_{th} \\ (max_{(x,y)} - min_{(x,y)}) > \gamma \end{cases} \\ \text{false, otherwise} \end{cases}, \tag{3.6}$$

where μ_{th}, σ^2_{th} and γ are the given thresholds for the mean, variance and range of the sample, respectively. By combining the mean, variance and range we are able to predict with a higher degree of certainty the presence of obstacles, using the thresholds to tune the detector to the general type of evident obstacles expected. For instance, we can use a higher variance and mean if we expect to be dealing with grown trees. On the other hand, we can lower both those values if the terrain is mostly free of tall obstacles and have mostly smaller rocks instead.

Fig. 3.8 shows the output from the evident obstacles detector in a forest scenario, where it is clear that the algorithm is considering every tree as an obstacle while discarding the tree tops, thus generating a traversable map that would otherwise be unattainable.



(a) Top view.

(b) Side view.

Figure 3.8: Representation of the evident obstacles detection stage output on a forestry scenario.

We implemented this functionality using a ROS node that re-uses all the possible code from the other stages, mainly the *terrain roughness estimation* stage.

3.7 Terrain Roughness Estimation

This module estimates the terrain roughness using the previously generated gradient map. In general, when considering a 3D pointcloud representing the environment, the gradient along its x and y coordinates provides very useful information about the structure of the environment.

The definition of mechanical effort from [46] is presented below:

$$\gamma = \|\nabla f(x, y)\| \cdot \cos\theta \tag{3.7}$$

where θ is the angle between the vector that connects the position of the robot with the position of the cell we are currently analyzing and the gradient vector in that cell.

As depicted in Fig. 3.9, the terrain roughness estimation module takes as input a gradient map (in this case from a hill that is equally steep from every angle) and fuses that information with the notion of mechanical effort (Eq. 3.7), generating an output that takes into consideration the position of the robot with relation to

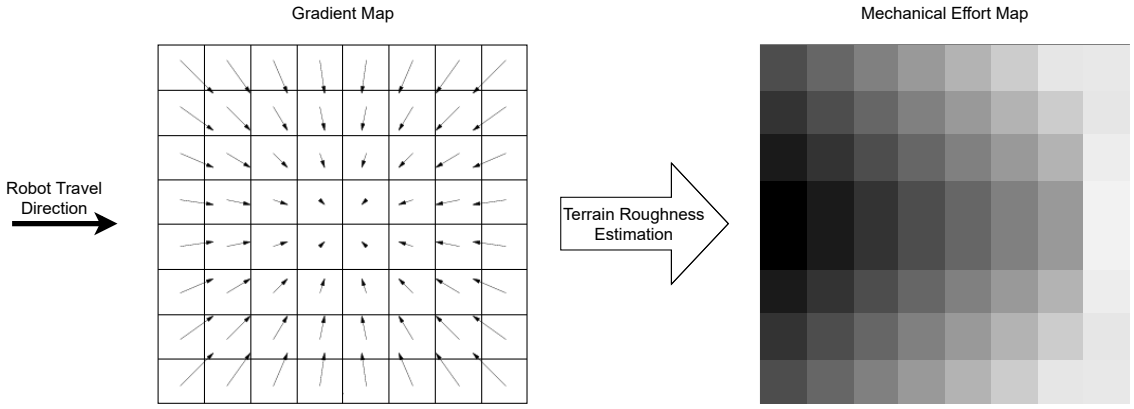


Figure 3.9: Illustration of the terrain roughness estimation output on a centered hill.

each cell of the gradient map, thus generating a map in which the traversability cost increases as the robot's direction of movement aligns with the gradient and vice-versa.

3.8 Data Merging

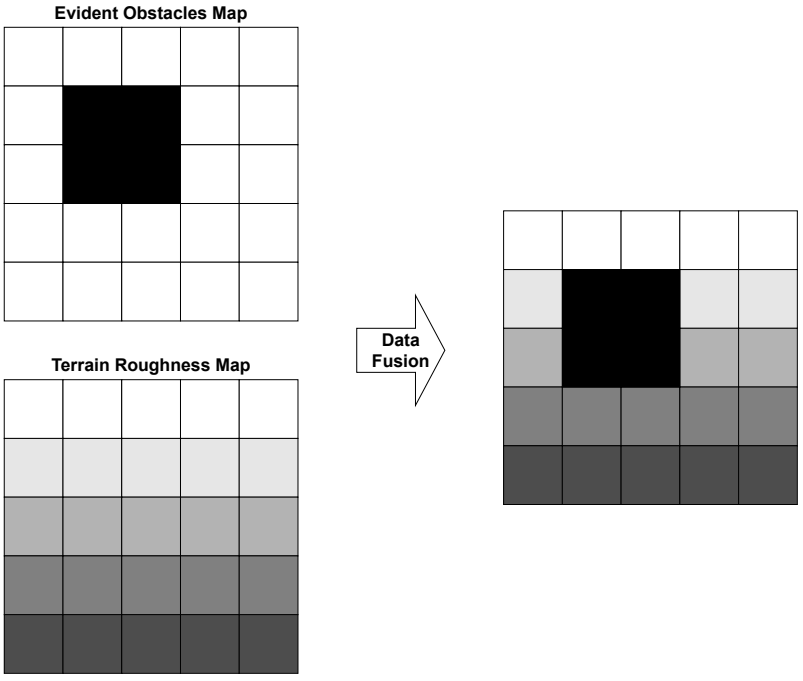


Figure 3.10: Illustration of the data fusion step.

This module outputs a single costmap that incorporates all the useful information contained on the *evident obstacles map* and the *terrain roughness map*, which are

the outputs of the modules described above. All the cells identified by the *evident obstacle detection* block are marked as non-traversable (100) in the final costmap, being that all the remaining ones are marked with an integer ranging from 0 to 99, where 50 represents a neutral cost of traversing (levelled terrain), 0 represents the lowest possible traversing cost (downhill slope that does not exceed the angular limits of the robot), and 99 the maximum possible cost of traversing a cell that is not an obstacle.

Fig. 3.10 represents this stage's output as described above, where we can clearly see how the costmap fusion is performed.

4 Experimental Evaluation

This chapter describes the experimental results of the tests performed to validate and verify the proposed method, as well as the performance metrics and experimental scenarios created that enable us to perform them. During these tests, we compare our approach to some of the approaches discussed in the state of the art analysis and we present a discussion around them.

4.1 Objectives of the Experiments

We have defined a set of tests and comparisons that are presented in this chapter and whose main objective is to accomplish the following goals:

1. Demonstrate that our work manages to create and execute paths with a lower energetic cost when compared to others.
2. Verify that our method outperforms the method in which it is based.
3. Show that our method can run in real time.

4.2 Performance Metrics

All of the raw data generated $(x, y, timestamp)$, as well as all of the metrics defined below are logged during the experiments and saved into logfiles in `.csv` format. In order to quantify the performance of our method and compare it against others, we have defined the following metrics:

Elapsed Time - The total time that it took for the robot to go from the starting point to the end goal.

Naturally, despite not being one of the main focus of our work in particular, it is important to compare the execution times of all techniques, even if to simply help differentiating two distinct approaches which are otherwise identical. Nevertheless, any technique that generates a theoretically perfect path according to every metric while taking a disproportional amount of time to do so, is not considered the most efficient one.

In general, a method should minimize the elapsed time without compromising other more important factors.

This metric helps to sustain objective number 2.

Travelled Distance - The total distance travelled by the robot from the starting point to the end goal.

This is an important metric, given that the travelled distance is naturally one of the most relevant factors when discussing fuel consumption and fuel efficiency. Generally speaking, a technique should be rewarded for reducing the travelled distance from the starting pose to the goal, with some exceptions such as not compromising/worsening some other important metric.

Using the error-free localization information provided by the simulator, the total travelled distance D is iteratively computed according to the following equation:

$$D = \sum_{k=1}^{N} \sqrt{(x_k - x_{k-1})^2 + (y_k - y_{k-1})^2 + (z_k - z_{k-1})^2}, \tag{4.1}$$

where x_τ, y_τ, z_τ are the robot's coordinate position in each of the axis (x,y,z), respectively, with τ representing the current (k) and previous ($k-1$) iterations, and N the total number of iterations.

This metric helps demonstrating objectives number 1 and 2.

Mean Map Generation Time - The time it takes to generate a new costmap from a pointcloud of N points.

For the purpose of this work, we consider that any approach runs in real time if a new map is generated before the robot reaches the end of the current one. This is clearly important, given that it represents the difference between the robot navigating known or unknown terrain.

We timed the execution of each iteration of our algorithm and saved that information along with the number of points in each map. After the execution of the trajectory, we compute the mean of all values and obtain the mean map generation time as well as the algorithm's capability of processing maps, measured in points per second.

This metric helps demonstrating objectives number 2 and 3.

Up Variation - The cumulative vertical distance travelled by the robot. In general, a technique should attempt to minimize this metric.

Each iteration of this metric is calculated according to

$$\beta_k^+ = \begin{cases} z_k - z_{k-1},\ z_k - z_{k-1} > 0 \\ 0, \text{ otherwise} \end{cases}, \tag{4.2}$$

being the total positive height variation given by

$$\beta^+ = \sum_{k=1}^{N} \beta_k^+. \tag{4.3}$$

This metric helps to demonstrate objective number 1.

Mean Effort - The mean effort that a technique faces while executing the given path. This metric is measured by averaging all the absolute pitch values, according to:

$$\Psi = \frac{\sum_{k=1}^{N} |\theta_k|}{N}, \tag{4.4}$$

where Ψ represents the mean effort, θ represents the pitch value, N represents the total number of iterations and k the current iteration.

Considering that this metric gives us an indication of the mean vertical travelling angle – a direct correlation with the effort associated with the chosen path – the candidate techniques are rewarded for minimizing it.

Naturally, this metric helps to demonstrate objective number 1.

Path Riskiness Index - The path riskiness index metric indicates how risky the travelled path is. In this context, we define a "risky" situation as one where the robot travels while exceeding any of its angular limits, and we quantify it as the percentage of time that the robot exceeded said limits in the travelled path. This is a very important metric, considering that the ultimate goal of this work is to enable a real robot to operate autonomously on highly demanding scenarios such as forests, therefore it must avoid dangerous portions of terrain with a high degree of certainty in order to maximize its autonomy and minimize the associated maintenance costs. According to the above considerations, the tested techniques should minimize this metric.

This metric helps demonstrating objective number 2.

Roll Danger Index - The percentage of roll-related risk that the robot takes during the execution of its path. The roll values are filtered with the following sigmoid function:

$$RDI = \frac{1}{1 + e^{-(0.5\phi - 13)}}, \tag{4.5}$$

where ϕ is a vector containing all the recorded roll values in degrees.

The purpose of filtering these values with a sigmoid function is to de-linearize the penalty curve of the robot's inclination. Considering the roll angle limit to be 35^{o}, it is almost equally safe to travel with between 5^{o}-10^{o} of inclination. However, the same 5^{o} difference between 25^{o}-30^{o} of inclination should not be taken as lightly, considering that the stability of the vehicle rapidly decreases when approaching its angular limits. A sigmoid filter adequately mimics this behavior, hence our choice of adopting it.

Pitch Danger Index - The percentage of pitch-related risk that the robot takes during the execution of its path. Similarly to the previous metric and according to the same reasons, the pitch values are also filtered with the following sigmoid function:

$$PDI = \frac{1}{1 + e^{-(0.25\theta - 6)}} \tag{4.6}$$

where θ is a vector containing all the recorded pitch values in degrees.

Failure Rate - We define the failure rate of a given algorithm as the percentage of times that its path planner aborts the whole mission because of one of the following reasons:

- It has not been able to follow the selected path (e.g. due to some internal error in the navigation stack).
- It has not been able to create a valid or feasible plan from the current pose to the targeted goal.
- The specified timeout for the mission has been exceeded.
- The robot rolled over.

This metric accounts for any situation that would lead to a failed run and marks it, though providing a solid measure of a system's reliability and robustness and helping to demonstrate objective number 2.

These metrics allow us to test our algorithm by means of a quantitative analysis, as well as to compare it to existing ones. Yet, there were still two main impositions that had to be addressed: the necessity of having a simulation tool where a given simulation could run N consecutive times (Section 4.3), and some way of extracting metrics from those experiments.

In order to collect the chosen metrics, a script was developed to listen to meaningful information on ROS topics, automatically compute every metric and log the raw data $(x, y, timestamp)$ and the calculated metrics.

It is important to note that for this work we do not consider the problem of estimating robot localization. For these results, we assume the error-free localization given by the 3D simulator used through its ROS interface.

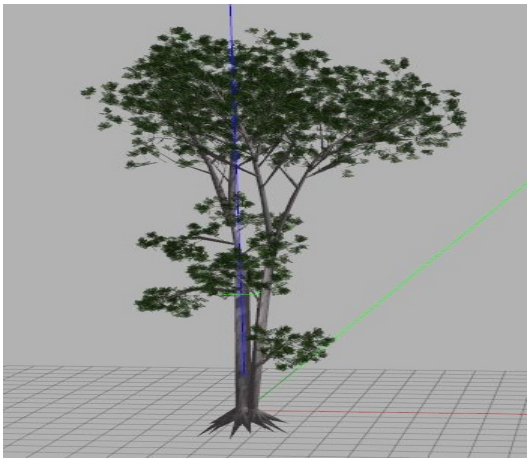
4.3 Simulation Environment

We created a 3D Gazebo simulation environment to perform the experiments and enable the comparison between the selected techniques. Gazebo is a 3D simulation tool that allows its users to create realistic simulations, with complex scenarios and a robust physics engine [10]. Among other features, such as its strong community, there is a ROS package[1] that has the interface between the simulated data and the ROS middleware implemented, which allows to simulate robots using ROS with a fraction of the effort. Fig 4.1b shows the ranger in a 3D Gazebo forest simulation.

The environment's terrain is created using the simulator's heightmap interpretor, that interprets a `.png` graphics file and translates it into a height mesh. In our scenarios, we have used two different terrains, shown in Fig. 4.1c and Fig. 4.1d, and one type of obstacle, a fully grown pine tree (Fig. 4.1a). Furthermore, the ranger (see Fig. 1.1) and the 16 channel LiDAR sensors used are simulated according to their technical characteristics, enabling us to obtain the sensor measurements required to perform the experiments.

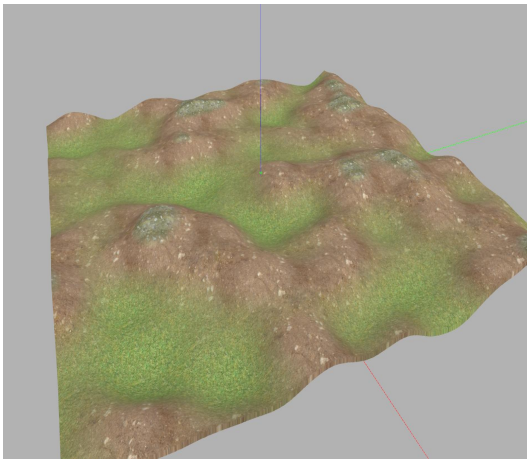
This simulation environment can be launched from the terminal - through the use of custom scripts – thus enabling us to perform autonomous experiments sequentially with N runs each. This is a particularly relevant characteristic as it allows us to place the robot anywhere in the workspace and give it any possible goal by changing the respective configuration files without the need to manually run consecutive experiments.

[1] http://wiki.ros.org/gazebo_ros

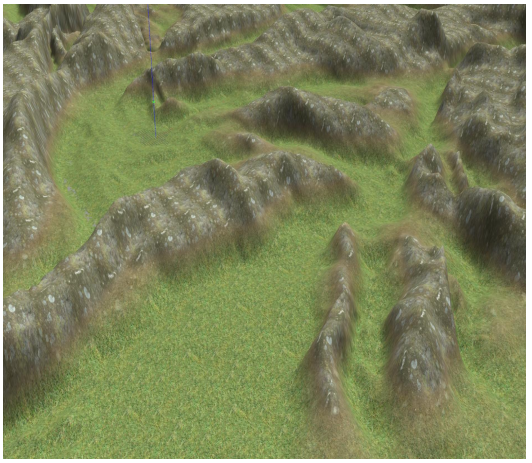


(a) Chosen tree model.

(b) Ranger in a forest scenario.



(c) First terrain.



(d) Second terrain.

Figure 4.1: Simulation models and terrains used.

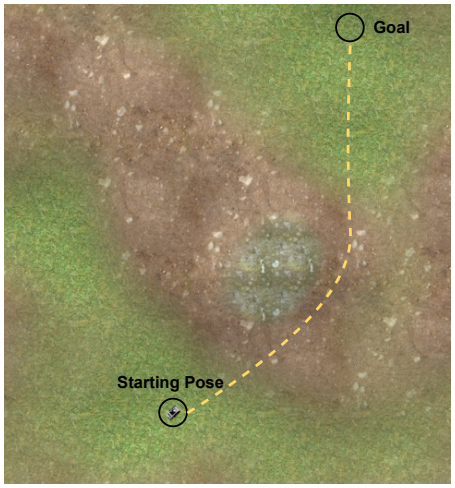
4.4 Scenarios

Fig. 4.2 illustrates Scenario 1, featuring slight hills, depressions and valleys with the starting pose and the goal being on opposite sides of a hill, as shown in Fig. 4.2b. Considering that this hill is mostly traversable without exceeding the angular limits of the robot, it allows for a myriad of different paths to be successfully executed. In Fig. 4.2a, we have represented in yellow what we found to be the approximate path followed by our technique when minimizing the mechanical effort and the path riskiness index while still regarding the travelled distance and, indirectly, the travel time. This path is represented to provide some insight into the reasoning behind the choice of each particular scenario.

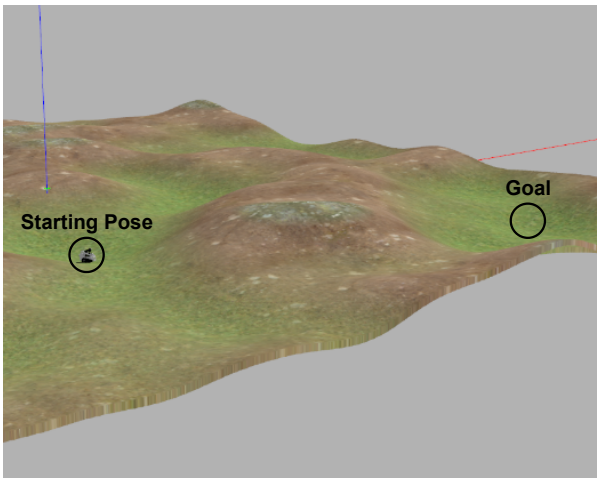
In Fig. 4.3 we have a representation of Scenario 2. The idea behind this scenario is similar to the first one, with one major difference: the hills in this scenario are much steeper, which means that the traversing cost across the hills will be significantly higher, and so will the respective risk. This distinct characteristic makes this an appropriate benchmarking scenario, rewarding path safety over travel distance, travel time and even mechanical effort, while naturally still taking them into account.

We also designed a second version of Scenario 2, which we call "Scenario 2 remapped" and consists of the same terrain and features, but with one major difference: the slopes partially exceed the angular limits of the robot. This slight variation is critical, considering that it marks the difference between steep slopes, and slopes that are mostly impossible to safely traverse.

Fig. 4.4 represents Scenario 3. This scenario has an extra layer of complexity when compared to the first two, considering that besides the hilly terrain it also has a forest of fully grown pine trees on it. This poses a double challenge to the candidate techniques, as they have to be able to safely navigate the trees without colliding with them while, hopefully, choosing an efficient path to do so.

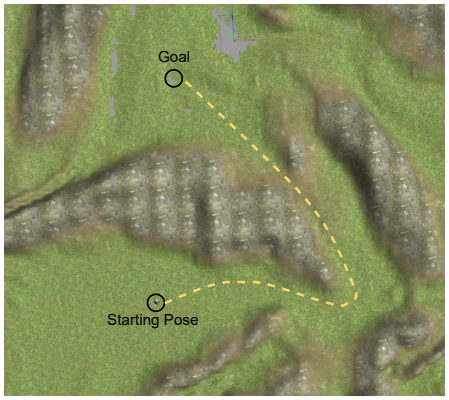


(a) Top view of Scenario 1.

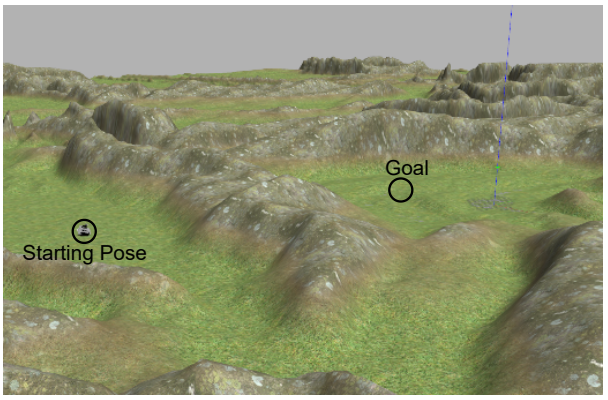


(b) Side view of Scenario 1.

Figure 4.2: First experimental scenario from two different perspectives, with starting pose, goal point and expected approximate path for our technique in yellow.

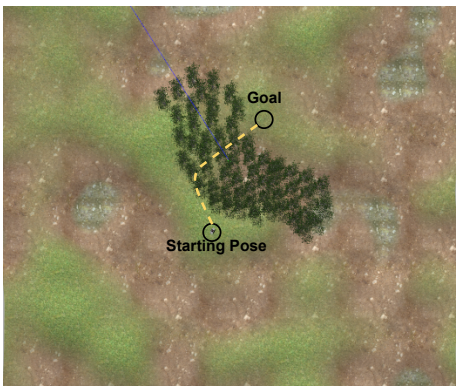


(a) Top view of Scenario 2.

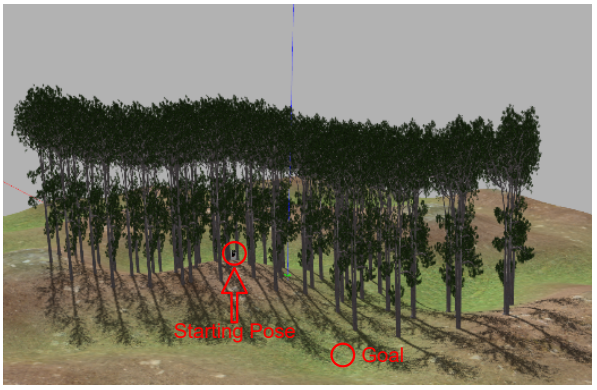


(b) Side view of Scenario 2.

Figure 4.3: Second experimental scenario from two different perspectives, with starting pose, goal point and expected approximate path for our technique in yellow.



(a) Top view of Scenario 3.



(b) Front view of Scenario 3.

Figure 4.4: Third experimental scenario from two different perspectives, with starting pose, goal point and expected approximate path for our technique in yellow.

4.5 Parameterization of the Compared Methods

This section describes the different techniques that are compared, as well as the respective choice of parameters to configure each one.

We compared our method with:

move_base[2]**,** which has been used with no observation sources and empty costmaps, always following a straight path to the goal and thus being referred to as **move_base_naive** from now on. This is an important method to use in our comparisons, considering that it establishes the baseline, i.e. the indicators that result from a straight path. With this in mind, it is expected that this technique highly struggles in cluttered environments, or any environment in general where there are obstacles in its path. On the other hand, in the experimental scenarios with no obstacles, its expected that this technique performs decently, generally achieving low values of elapsed time considering that it always execute the straightest path to the goal.

System from [46], which uses raw sensory data as input to a pipeline that populates a 2D costmap using the mechanical effort concept proposed in [9], resorting to a median-based interpolation to fill the cells that contain no information. The

[2]http://wiki.ros.org/move_base

generated map is fed into move_base's local planner, which drives the robot accordingly. This technique is generally inefficient, considering that its known horizon is relatively small, and it struggles with obstacles near the robot, considering that the interpolation technique cannot accurately predict them. Having the above considerations in mind, it should come at no surprise that this method may produce poor results.

move_base_flex[3] **[47],** which adheres to move_base's local and global path planners' interface. However, it was designed to provide richer feedback to the user, which is valuable information for debugging, and it allows the implementation of custom behavior trees to define the platform's actions more precisely and robustly, as well as navigation with meshes instead of costmaps. Unfortunately, this method has not yielded consistent results, either by being unable to perform in some scenarios or by generating results that were too similar to other techniques, so it will not be considered in any of the comparisons.

The systems have been configured according to Table 4.1, where the parameters which are not mentioned have been kept default or are not relevant to the current discussion. As much as possible, all the systems have been configured with the same values so that the comparisons are as fair as possible. However, the system from [46] has been used following the author's own configuration, considering that, in principle, they should be the most suitable for that particular system.

[3]http://wiki.ros.org/move_base_flex

Table 4.1: Parameters used for the different techniques.

Parameters	Our System	System from [46]	move_base_naive	move_base_flex
Global Planner	A*	Dijkstra	A*	A*
Local Planner	Trajectory Rollout	Trajectory Rollout	Trajectory Rollout	Trajectory Rollout
planner_frequency	0	0	0	0
controller_frequency	5	3	5	5
allow_unknown	false	true	false	false
use_dijkstra	false	true	false	false
use_quadratic	false	true	false	false
use_grid_path	false	false	false	false
old_navfn_behavior	false	false	false	false
visualize_potential	false	false	false	false
cost_factor	1	3	1	1
neutral_cost	50	50	50	50
lethal_cost	100	253	100	100
acc_lim_x	5	0.4	5	5
acc_lim_y	5	0.4	5	5
acc_lim_theta	5	0.5	5	5
max_vel_x	1.5	0.9	1.5	1.5
min_vel_x	-1.5	0.3	-1.5	-1.5
max_vel_theta	1.5	1	1.5	1.5
min_vel_theta	-1.5	-1	-1.5	-1.5
min_in_place_vel_theta	0.314	1	0.314	0.314
holonomic_robot	false	false	false	false
escape_vel	-0.5	-0.1	-0.5	-0.5
yaw_goal_tolerance	6.2832	6.2832	6.2832	6.2832
xy_goal_tolerance	0.5	1	0.5	0.5
sim_time	2	2	2	2
meter_scoring	true	true	true	true
path_distance_bias	0.4	0.08	0.4	0.4
goal_distance_bias	0.1	0.05	0.1	0.1
plugins	costmap_2d::StaticLayer	SpatioTemporalVoxelLayer costmap_2d::InfationLayer	None	costmap_2d::StaticLayer

4.6 Results and Discussion

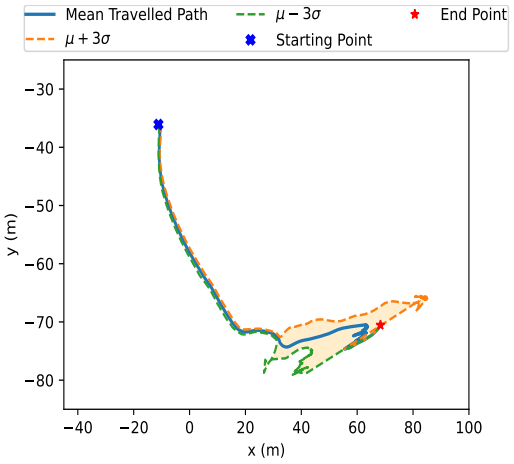
In this section, the results of the undertaken experiments are presented, both qualitatively and quantitatively between the compared methods. Each of the experiments have been executed 50 times so that we could perform a statistical analysis of the obtained results.

We would like to mention some aspects that apply to the remaining of the discussion, namely:

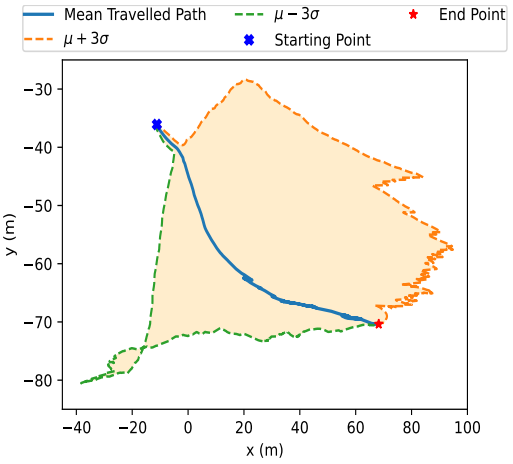
- In the results tables we highlight in **bold** the best value obtained for each metric.
- As mentioned in Section 4.5, `move_base_flex` has not yielded consistent results, so it will not be considered in the following comparisons.

The results of the first experiment performed, in Scenario 1 (Fig. 4.2), are presented in Table 4.2. As stated before in Section 4.4, this is the least demanding scenario, with no obstacles and slight hills that are generally traversable and with little to no risk for the robot.

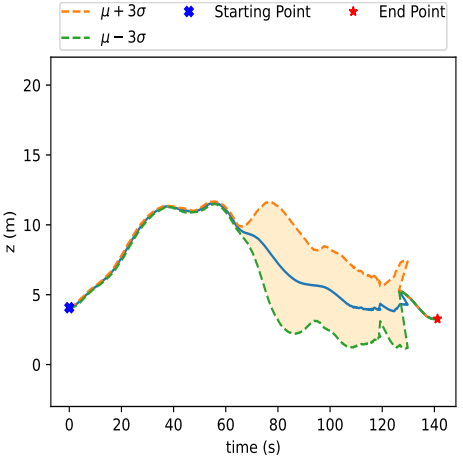
Even though the chosen paths for Scenario 1 are considerably different (as illustrated in Fig. 4.5), the differences in the metrics from all systems, with the exception of the elapsed time, are relatively small. This is an expected result in light of the above considerations regarding the complexity of the scenario. Our system effectively managed to minimize the first 4 metrics presented in Table 4.2, which are the ones that we are prioritizing in this work, while only slightly sacrificing the total travelled distance, the elapsed time and roll danger. Regarding the roll danger index, it is important to state that, although *move_base_naive* had a result that was much lower than both other techniques, all the values are negligible, given that even the worst performing system only travelled, on average, with an inclination of about 1.9% of the limit value.



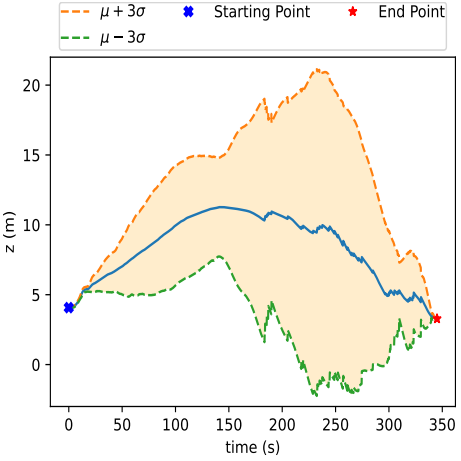
(a) 50 trajectories projected on the X-Y plane with our system.



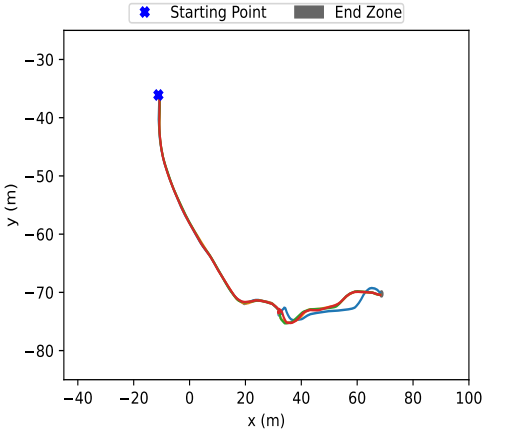
(b) 50 trajectories projected on the X-Y plane with the system from [46].



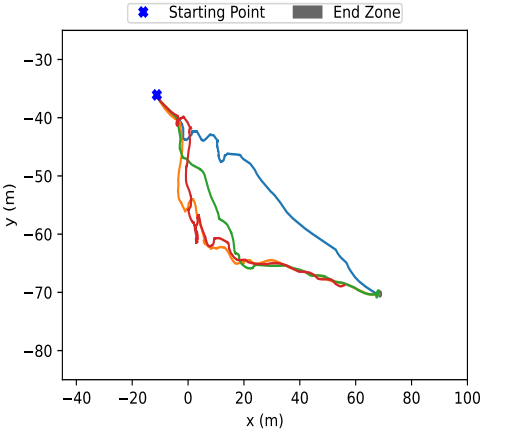
(c) 50 trajectories of height as a function of time with our system.



(d) 50 trajectories of height as a function of time with the system from [46].



(e) Robot's 4 hand picked trajectories projected on the X-Y plane with our system.



(f) Robot's 4 hand picked trajectories projected on the X-Y plane with the system from [46].

Figure 4.5: Resulting plots from 50 runs in Scenario 1 (Fig. 4.2) with our system in the left, and the system from [46] in the right.

Table 4.2: Results from 50 runs in Scenario 1 (Fig. 4.2), where MMGT stands for "Mean Map Generation Time" and is measured in (seconds taken @ points in the map (points/second)).

Metrics	System from [46]	move_base_naive	Our System
Failure Rate (%)	10.000	**8.000**	**8.000**
Mean Effort (º)	11.701 ± 1.064	12.520 ± 0.047	$\mathbf{10.202 \pm 0.423}$
Pitch Danger Index (%)	11.290 ± 2.863	13.758 ± 0.136	$\mathbf{8.796 \pm 1.141}$
Up Variation (m)	10.433 ± 0.938	9.058 ± 0.017	$\mathbf{8.666 \pm 0.559}$
Travelled Distance (m)	112.839 ± 8.005	$\mathbf{89.645 \pm 0.053}$	106.388 ± 4.073
Path Riskiness Index (%)	0.112 ± 0.732	$\mathbf{0.000 \pm 0.000}$	0.068 ± 0.201
Roll Danger Index (%)	1.903 ± 1.428	$\mathbf{0.016 \pm 0.001}$	1.888 ± 0.652
MMGT (s @ pts (pts/sec))	11.3 @ 37.6k (3.3k)	N/A	15.4 @ 210k (13.6k)
Elapsed Time (s)	279.612 ± 49.750	$\mathbf{87.031 \pm 23.229}$	111.326 ± 12.938

In Table 4.3 we have the results of the experiment comparing the chosen techniques in Scenario 2 (Fig. 4.3), where the computed values for every metric are registered and respectively plotted in Fig. 4.7.

As shown in Fig. 4.7, our system takes a very different approach at this harder scenario. Instead of trying to traverse straight to the goal, which would represent a shorter elapsed time and travel distance at the cost of major risk (especially pitch related risk) and significantly higher mechanical effort, it chooses the more conservative and levelled path available, which is around the hill. By doing so, our system was able to very significantly reduce the effort associated with that trajectory and, per the argument regarding the relationship between terrain gradient and fuel consumption discussed in Section 2.3, potentially the fuel consumption.

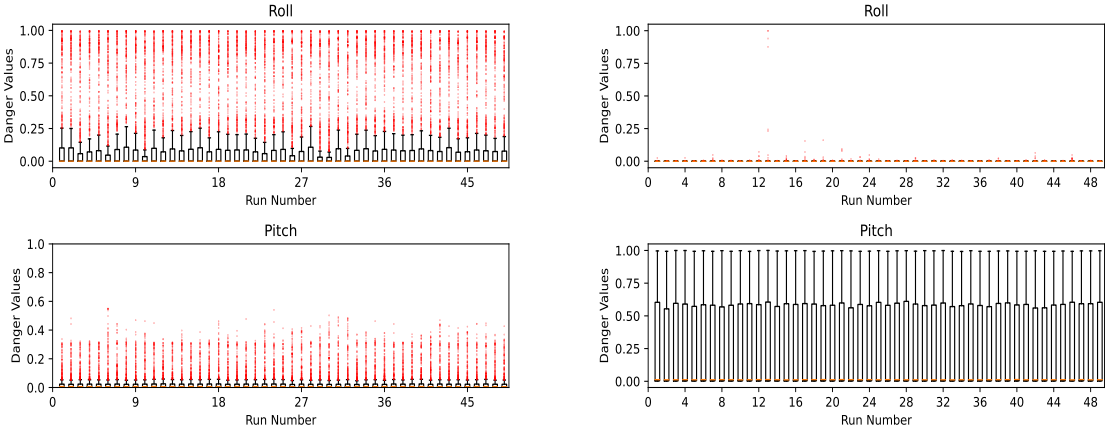
In Fig. 4.7a, we can observe that on a small portion of the trajectory performed by our system the variance is a little higher, where we can isolate two different path choices. This is due to the fact that both choices make sense in the context of the optimization problem at hand, one being slightly farther away and the other having a slightly lower height variation, but both having a very similar mechanical effort associated.

It is also worthy to note that the pitch danger index is significantly lower than both other techniques, while the roll danger index is not that low, specially when compared to the `move_base_naive` approach. This is, however, easily justifiable given that while the robot travels around the hill, it tries to minimize the travelled distance without incurring in excessive risk, thus travelling on the side slopes for the majority of the path, due to the morphology of the terrain in that area. Furthermore, even doing so, it only travels on average at 12.8% the maximum inclination allowed, while the system from [46] travels at a considerably higher 18.6%. This happens because [46]'s system detects the presence of a significant slope ahead, but is unable to find a better path due to the fact that its sensory horizon is short, so it tries to travel sideways on the slope and find a better path, which is not an ideal behavior considering that it significantly increases both danger and riskiness indexes and the elapsed time.

Fig. 4.6 represents the statistical distribution of the collected roll and pitch danger indexes on Scenario 2. Clearly, Fig. 4.6a shows that, although the roll values generated by our technique are indeed substantially higher than with move_base_naive, they are still negligible in the sense that their distribution is generally low, well below the 50% risk. On the other hand, the same does not apply to the distribution of values from the experiment with move_base_naive, considering that a significant portion of the distribution consistently returned danger values above 50%, while a non-negligible portion reaches all the way up to 100%.

It is noteworthy that, although there is a major visual impact caused by outliers, especially in the experiment with our system (Fig. 4.6a), they represent a negligible slice of the distribution, not exceeding a few percentage points at most.

Table 4.4 shows the results from running the same experiment in the remapped Scenario 2 (see Section 4.4). Here we can see that by slightly increasing the slope angles, the other systems perform drastically worse, while ours maintains the results without much change. The large failure rates from the other two techniques come at no surprise, considering that this scenario was specifically designed to have this effect with any technique that chooses to ignore its large hills, reinforcing the importance



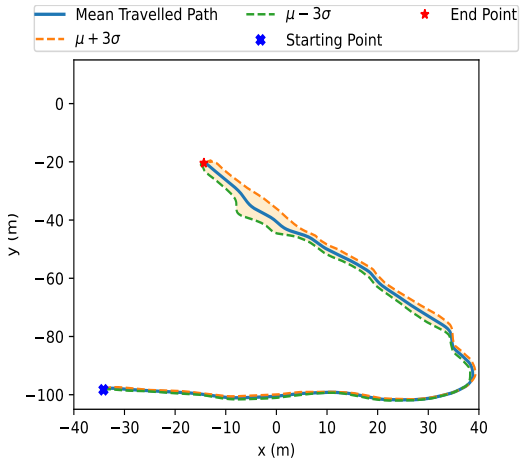
(a) Our system.

(b) move_base_naive.

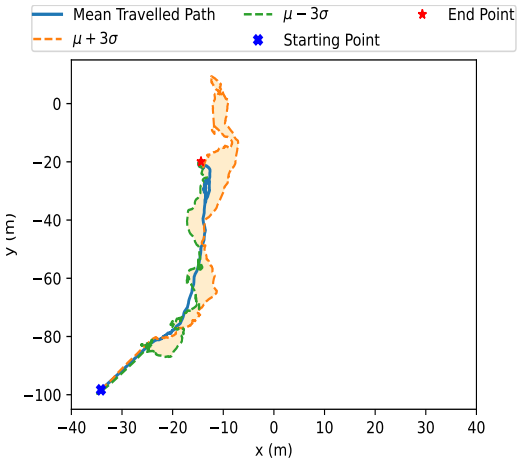
Figure 4.6: Resulting boxplots from 50 runs in Scenario 2 (Fig. 4.3).

Table 4.3: Results from 50 runs in Scenario 2 (Fig. 4.3), where MMGT stands for "Mean Map Generation Time" and is measured in (seconds taken @ points in the map (points/second)).

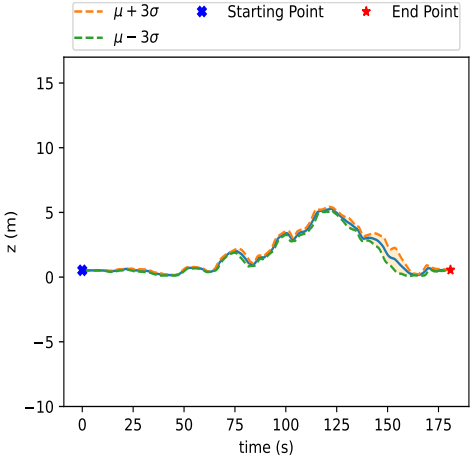
Metrics	System from [46]	move_base_naive	Our System
Failure Rate (%)	84.000	**2.000**	**2.000**
Mean Effort (º)	18.661 ± 1.828	12.805 ± 0.205	$\mathbf{5.601 \pm 0.060}$
Pitch Danger Index (%)	38.592 ± 4.892	25.406 ± 0.335	$\mathbf{2.814 \pm 0.142}$
Up Variation (m)	15.240 ± 1.886	8.877 ± 0.036	$\mathbf{8.183 \pm 0.088}$
Travelled Distance (m)	112.387 ± 8.066	$\mathbf{84.958 \pm 0.068}$	171.887 ± 0.343
Path Riskiness Index (%)	4.291 ± 4.094	$\mathbf{0.517 \pm 0.650}$	0.932 ± 0.212
Roll Danger Index (%)	26.580 ± 4.648	$\mathbf{0.042 \pm 0.137}$	12.843 ± 0.695
MMGT (s @ pts (pts/sec))	8.2 @ 40.1k (4.9k)	N/A	18.3 @ 287k (15.7k)
Elapsed Time (s)	343.582 ± 14.886	$\mathbf{87.014 \pm 12.689}$	174.368 ± 4.184



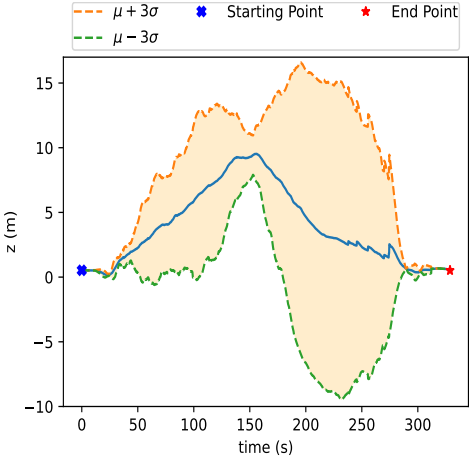
(a) 50 trajectories projected on the X-Y plane with our system.



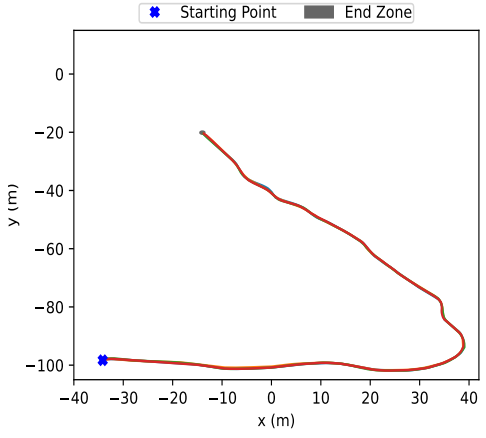
(b) 50 trajectories projected on the X-Y plane with the system from [46].



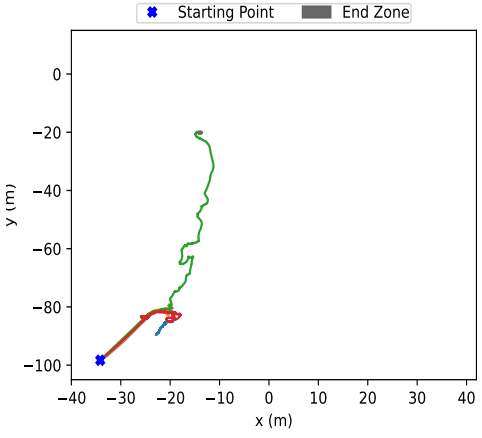
(c) 50 trajectories of height as a function of time with our system.



(d) 50 trajectories of height as a function of time with the system from [46].



(e) Robot's 4 hand picked trajectories projected on the X-Y plane with our system.



(f) Robot's 4 hand picked trajectories projected on the X-Y plane with the system from [46].

Figure 4.7: Resulting plots from 50 runs in Scenario 2 (Fig. 4.3) with our system in the left, and the system from [46] in the right.

of avoiding the steepest sections of terrain and ultimately showing the robustness and reliability of our system, while also supporting objectives 1 and 2.

Lastly, Table 4.5 presents the results of the comparison between the chosen techniques in Scenario 3 (Fig. 4.4), arguably the most demanding scenario that we used considering that not only the robot must avoid every tree in its path, but it should also choose the path with the lowest effort associated.

At first, there are a few things that jump out straight away, such as the fact that the path riskiness index from our system is 0% and the substantial difference in execution times and failure rates. The particular paths chosen by our technique in this scenario does not have a single point where the inclination of the terrain is such that the robot exceeds the roll and pitch limits, as can be partially seen in Fig. 4.4b, thus supporting the plausibility of this particular result.

Our system[4] greatly outperformed the remaining ones in every single metric except in the travelled distance, which was slightly higher. Note, however, how that little increase in travel distance produced a significant difference in the mean effort and vertical variation, which are the main variables that we take into consideration.

As stated before, the consistency and robustness of our technique is latent in the considerable difference in all of the results, and especially in the failure rate and mean effort.

In all the experiments and results presented above, our system significantly outperformed the system from [46] with regards to the "Mean Map Generation Time" metric, which is an expected result according to Section 3.3.

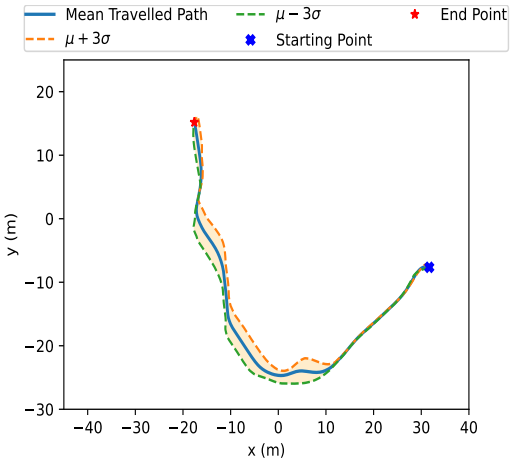
[4]There is a video demonstrating our system in action on Scenario 3, available here: `https://drive.google.com/drive/folders/1vbsacOTLBlUhMT2tEWYp-1_-D22ncDUS?usp=sharing`

Table 4.4: Results from 50 runs in Scenario 2 remapped, where MMGT stands for "Mean Map Generation Time" and is measured in (seconds taken @ points in the map (points/second)).

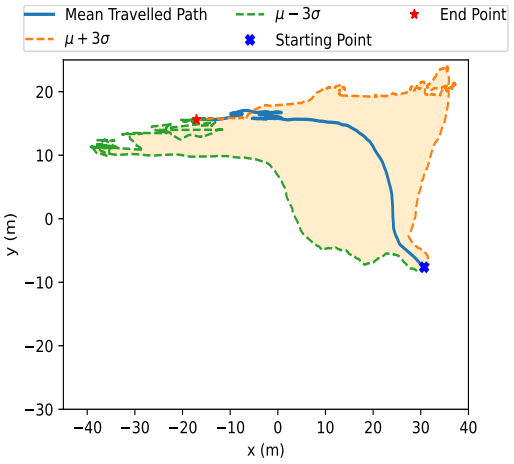
Metrics	System from [46]	move_base_naive	Our System
Failure Rate (%)	100.000	90.000	**10.000**
Mean Effort (º)	N/A	17.172 ± 0.902	$\mathbf{7.004 \pm 0.301}$
Pitch Danger Index (%)	N/A	32.131 ± 1.199	$\mathbf{5.807 \pm 0.653}$
Up Variation (m)	N/A	11.586 ± 0.050	$\mathbf{10.319 \pm 0.476}$
Travelled Distance (m)	N/A	$\mathbf{87.873 \pm 0.399}$	175.689 ± 1.894
Path Riskiness Index (%)	N/A	12.558 ± 0.853	$\mathbf{2.839 \pm 1.134}$
Roll Danger Index (%)	N/A	$\mathbf{0.807 \pm 1.413}$	12.788 ± 2.581
MMGT (s @ pts (pts/sec))	8.9 @ 40.5k (4.6k)	N/A	18.3 @ 287k (15.7k)
Elapsed Time (s)	N/A	$\mathbf{25.299 \pm 2.510}$	173.037 ± 31.038

Table 4.5: Results from 50 runs in Scenario 3 (Fig. 4.4), where MMGT stands for "Mean Map Generation Time" and is measured in (seconds taken @ points in the map (points/second)).

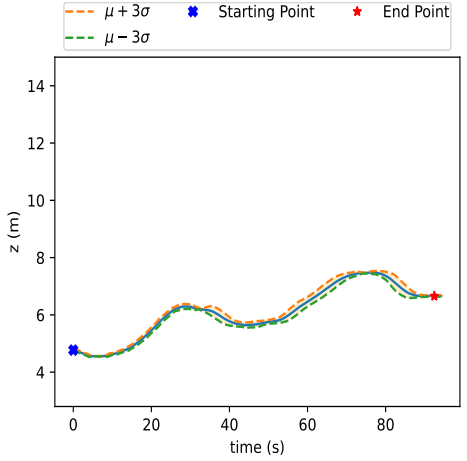
Metrics	System from [46]	move_base_naive	Our System
Failure Rate (%)	58.000	92.000	**6.000**
Mean Effort (º)	11.024 ± 3.500	15.188 ± 0.100	$\mathbf{3.595 \pm 0.066}$
Pitch Danger Index (%)	9.326 ± 8.016	23.300 ± 0.565	$\mathbf{0.794 \pm 0.015}$
Up Variation (m)	7.879 ± 0.496	7.035 ± 0.066	$\mathbf{3.714 \pm 0.045}$
Travelled Distance (m)	80.226 ± 9.643	$\mathbf{58.299 \pm 0.497}$	85.586 ± 0.493
Path Riskiness Index (%)	1.353 ± 5.998	0.767 ± 0.701	$\mathbf{0.000 \pm 0.000}$
Roll Danger Index (%)	0.297 ± 0.297	0.909 ± 0.469	$\mathbf{0.020 \pm 0.001}$
MMGT (s @ pts (pts/sec))	12.1 @ 42.4k (3.5k)	N/A	20.2 @ 358k (17.7k)
Elapsed Time (s)	340.362 ± 123.149	182.514 ± 193.306	$\mathbf{88.266 \pm 12.011}$



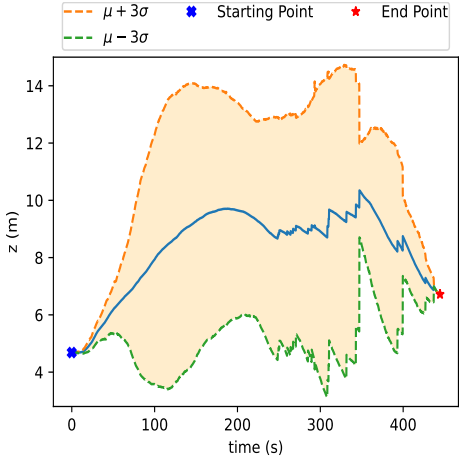
(a) 50 trajectories projected on the X-Y plane with our system.



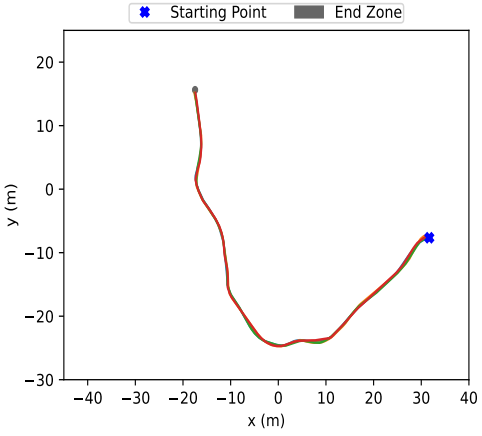
(b) 50 trajectories projected on the X-Y plane with the system from [46].



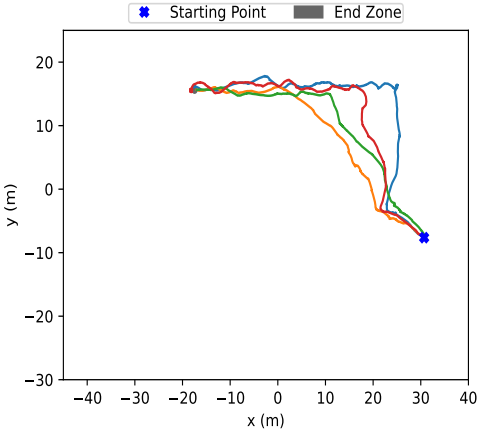
(c) 50 trajectories of height as a function of time with our system.



(d) 50 trajectories of height as a function of time with the system from [46].



(e) Robot's 4 hand picked trajectories projected on the X-Y plane with our system.



(f) Robot's 4 hand picked trajectories projected on the X-Y plane with the system from [46].

Figure 4.8: Resulting plots from 50 runs in Scenario 3 (Fig. 4.4) with our system in the left, and the system from [46] in the right.

5 Conclusion

This work proposes a novel traversability analysis and path planning technique based on the mechanical effort concept introduced in [9]. This technique categorizes terrain according to the effort required to traverse it, while identifying key evident obstacles, consequently generating efficient paths that avoid obstacles and major hills and thus potentially minimizing fuel consumption. A concurrent pipeline of lethal obstacles detection and terrain roughness estimation has also been implemented and the whole system has been optimized to execute in real time while performing a global analysis. This potentially allows a robot to plan in real-time far beyond its observable range if given an *a priori* map of the environment. Finally, the implemented system has been tested against other methods in a 3D realistic simulation engine, yielding very positive results and proving to be a strong competitor against other state of the art techniques.

Regarding the success of this work, we have accomplished the experimental objectives proposed in Section 4.1, we present a contribution to the first identified research gap (Section 2.4) while also successfully addressing one of [46]'s stated weaknesses, as shown in Section 3.3, which suggested that the map generation frequency should be improved.

As with any scientific research one can think of, there are many lessons learned and room to improve upon our work. As possible paths forward and hypothesis to pursue, we would like to mention:

1. Even though our work already manages to run in real time, it would still be a very interesting idea to further improve the efficiency of the algorithm, enabling larger costmaps to be used and/or faster throughput. This can be achieved

by vectorizing the calculations, by using multi-threads or multi-processes or by implementing a caching/incremental calculation system so that the map is not re-calculated at every iteration.

2. The system has been designed to have a fully deliberative behavior. This, of course, leaves room for improvement via the addition of a reactive module to enable the robot to detect and evade dynamic entities in real time.

3. Up to this point, the problem of having the mulching tool attached has not been taken into account. Considering that this tool partially occludes the sensors' field of view and changes the footprint of the platform depending on its current configuration, it would be interesting to model the tool and devise a strategy to intelligently operate it.

4. Test the developed technique on the real robot, considering that up to this stage it was just tested in simulation environments.

5. Create a fuel consumption model of the Ranger and tune/design a strategy to minimize it.

6. No system, including ours, explicitly takes into account the inclination limits of the robot, so they will take no action to prevent the robot from exceeding them. As such, the system can be enhanced to take this information into consideration, enabling it to avoid dangerous paths with a higher certainty.

7. Change the evident obstacles detector so that its thresholds can be autonomously inferred from the perceived environment and tuned to match the morphology of the potential obstacles.

6 Bibliography

[1] David Portugal, Maria Eduarda Andrada, André G Araújo, Micael S Couceiro, and João Filipe Ferreira. Ros integration of an instrumented bobcat t190 for the semfire project. In *Robot Operating System (ROS)*, pages 87–119. Springer, 2021.

[2] Micael S. Couceiro, David Portugal, Joao F. Ferreira, and Rui P. Rocha. SEMFIRE: Towards a new generation of forestry maintenance multi-robot systems. *Proceedings of the 2019 IEEE/SICE International Symposium on System Integration, SII 2019*, pages 270–276, 2019.

[3] Abbe Mowshowitz, Ayumu Tominaga, and Eiji Hayashi. Robot navigation in forest management. *Journal of Robotics and Mechatronics*, 30(2):223–230, 2018.

[4] Ayumu Tominaga, Hayashi Eiji, and Abbe Mowshowitz. Development of navigation system in field robot for forest management. *Proceedings - 2018 Joint 10th International Conference on Soft Computing and Intelligent Systems and 19th International Symposium on Advanced Intelligent Systems, SCIS-ISIS 2018*, pages 1142–1147, 2018.

[5] Nagham Shalal, Tobias Low, Cheryl McCarthy, and Nigel Hancock. Orchard mapping and mobile robot localisation using on-board camera and laser scanner data fusion–part b: Mapping and localisation. *Computers and Electronics in Agriculture*, 119:267–278, 2015.

[6] A. Linz, A. Ruckelshausen, E. Wunder, and J. Hertzberg. Autonomous service robots for orchards and vineyards: 3D simulation environment of multi sensor-

based navigation and applications. *12th International Conference on Precison Agriculture, ISPA International Society of Precision Agriculture*, 38(1):13, 2014.

[7] Keun Ha Choi, Sang Kwon Han, Sang Hoon Han, Kwang-Ho Park, Kyung-Soo Kim, and Soohyun Kim. Morphology-based guidance line extraction for an autonomous weeding robot in paddy fields. *Computers and Electronics in Agriculture*, 113:266–274, 2015.

[8] Tijmen Bakker, Kees van Asselt, Jan Bontsema, Joachim Müller, and Gerrit van Straten. An autonomous weeding robot for organic farming. In Peter Corke and Salah Sukkariah, editors, *Field and Service Robotics*, pages 579–590, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[9] D. Lourenço, J. F. Ferreira, and D. Portugal. 3D Local Planning for a Forestry UGV based on Terrain Gradient and Mechanical Effort. *In Proc. of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2020), Workshop on Perception, Planning and Mobility in Forestry Robotics (WPPMFR 2020), Las Vegas, NV, USA, Oct 25-29*, 2020.

[10] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004.

[11] Kumar Malu and Jharna Majumdar. Kinematics, Localization and Control of Differential Drive Mobile Robot. *Global Journal of Researches in Engineering*, 14(1):1–8, 2014.

[12] Tom Clynes. *Exclusive: Laser Scans Reveal Maya 'Megalopolis' Below Guatemalan Jungle*, 2018. `https://www.nationalgeographic.com/news/2018/02/maya-laser-lidar-guatemala-pacunam/`.

[13] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[14] Alif Ridzuan Khairuddin, Mohamad Shukor Talib, and Habibollah Haron. Review on simultaneous localization and mapping (SLAM). *Proceedings - 5th IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2015*, (November):85–90, 2016.

[15] Sebastian Thrun. Learning Maps for Indoor Mobile Robot Navigation. *Science*, 99(April):21–71, 1996.

[16] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[17] Alessandro Gasparetto, Paolo Boscariol, Albano Lanzutti, and Renato Vidoni. *Path Planning and Trajectory Planning Algorithms: A General Overview*, pages 3–27. Springer International Publishing, Cham, 2015.

[18] Liang Yang, Juntong Qi, Dalei Song, Jizhong Xiao, Jianda Han, and Yong Xia. Survey of Robot 3D Path Planning Algorithms. *Journal of Control Science and Engineering*, 2016, 2016.

[19] Lim Chee Wang, Lim Ser Yong, and Marcelo H. Ang. Hybrid of global path planning and local navigation implemented on a mobile robot in indoor environment. *IEEE International Symposium on Intelligent Control - Proceedings*, pages 821–826, 2002.

[20] Cang Ye. Navigating a mobile robot by a traversability field histogram. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(2):361–372, 2007.

[21] Cang Ye, Johann Borenstein, and Ann Arbor. T-transformation: Traversability Analysis for Navigation on Rugged Terrain. *International Society for Optics and Photonics*, pages 12–16, 2004.

[22] Chen Wang and Jian Mao. Summary of AGV path planning. *2019 IEEE 3rd International Conference on Electronic Information Technology and Computer Engineering, EITCE 2019*, pages 332–335, 2019.

[23] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1179–1187, 1989.

[24] ChaoChun Yuan, Yue Wei, Jie Shen, Long Chen, Youguo He, Shuofeng Weng, and Tong Wang. Research on path planning based on new fusion algorithm for autonomous vehicle. *International Journal of Advanced Robotic Systems*, 17(3):1729881420911235, 2020.

[25] Sining Yang, Shaowu Yang, and Xiaodong Yi. An Efficient Spatial Representation for Path Planning of Ground Robots in 3D Environments. *IEEE Access*, 6:41539–41550, 2018.

[26] Fabio Ruetz, Emili Hernández, Mark Pfeiffer, Helen Oleynikova, Mark Cox, Thomas Lowe, and Paulo Borges. OVPC Mesh: 3D Free-space Representation for Local Ground Vehicle Navigation. *arXiv*, pages 8648–8654, 2018.

[27] Péter Fankhauser, Michael Bloesch, and Marco Hutter. Probabilistic terrain mapping for mobile robots with uncertain localization. *IEEE Robotics and Automation Letters*, 3(4):3019–3026, 2018.

[28] Pablo Marin-Plaza, Ahmed Hussein, David Martin, and Arturo De La Escalera. Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles. *Journal of Advanced Transportation*, 2018, 2018.

[29] Sebastian Pütz, Thomas Wiemann, Jochen Sprickerhof, and Joachim Hertzberg. 3D Navigation Mesh Generation for Path Planning in Uneven Terrain. *IFAC-PapersOnLine*, 49(15):212–217, 2016.

[30] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[31] Kengo Usui. Data augmentation using image-to-image translation for detecting forest strip roads based on deep learning. *International Journal of Forest Engineering*, 00(00):1–10, 2020.

[32] Chaoqun Wang, Jiankun Wang, Chenming Li, Danny Ho, Jiyu Cheng, Tingfang Yan, Lili Meng, and Max Q.H. Meng. Safe and robust mobile robot navigation in uneven indoor environments. *Sensors (Switzerland)*, 19(13):1–20, 2019.

[33] Lei Tai, Shaohua Li, and Ming Liu. A deep-network solution towards model-less obstacle avoidance. *IEEE International Conference on Intelligent Robots and Systems*, 2016-Novem:2759–2764, 2016.

[34] J Borenstein and Yehuda Koren. The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, 1991.

[35] Niall Ormahony, Sean Campbell, Lenka Krpalkova, Daniel Riordan, Joseph Walsh, Aidan Murphy, and Conor Ryan. Deep Learning for Visual Navigation of Unmanned Ground Vehicles : A review. *29th Irish Signals and Systems Conference, ISSC 2018*, 2018.

[36] Ayanna Howard, Edward Tunstel, Dean Edwards, and Alan Carlson. Enhancing fuzzy robot navigation systems by mimicking human visual perception of natural terrain traversability. *Annual Conference of the North American Fuzzy Information Processing Society - NAFIPS*, 1:7–12, 2001.

[37] Assad Alam. *Fuel-efficient heavy-duty vehicle platooning. TRITA-EE 2014:027.* 2014.

[38] Yu Chen Lin and Ha Ly Thi Nguyen. Development of an eco-cruise control system based on digital topographical data. *Inventions*, 1(3):1–16, 2016.

[39] Min Zhou, Hui Jin, and Feng Ding. Minimizing vehicle fuel consumption on hilly roads based on dynamic programming. *Advances in Mechanical Engineering*, 9(5):1–8, 2017.

[40] Payman Shakouri, Andrzej Ordys, Paul Darnell, and Peter Kavanagh. Fuel efficiency by coasting in the vehicle. *International Journal of Vehicular Technology*, 2013, 2013.

[41] Georgios Fontaras, Nikiforos Georgios Zacharof, and Biagio Ciuffo. Fuel consumption and CO2 emissions from passenger cars in Europe – Laboratory versus real-world emissions. *Progress in Energy and Combustion Science*, 60:97–131, 2017.

[42] Gunnar Svenson and Dag Fjeld. The influence of road characteristics on fuel consumption for logging trucks. *HVTT12: 12th International Symposium on Heavy Vehicle Transport Technology*, 31(5):526–536, 2012.

[43] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. The office marathon: Robust navigation in an indoor office environment. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 300–307, 2010.

[44] Md Arafat Hossain and Israt Ferdous. Autonomous robot path planning in dynamic environment using a new optimization technique inspired by bacterial foraging technique. *Robotics and Autonomous Systems*, 64:137–141, 2015.

[45] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[46] Dora Lourenço. 3D Navigation for Ground Robots in Forestry Applications, MSc Thesis, DEEC, FCTUC. `https://eg.uc.pt/handle/10316/92225`.

[47] Sebastian Pütz, Jorge Santos Simón, and Joachim Hertzberg. Move Base Flex: A highly flexible navigation framework for mobile robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2018. Software available at `https://github.com/magazino/move_base_flex`.