Investigating 2.5D path-planning methods for autonomous mobile robots in complex unstructured off-road scenarios*

Andre Koczka[1], and Gerald Steinbauer-Wagner[1]

***Abstract*— Most of the existing literature focuses on path planning in 2D, where the 3D world is converted to a 2D grid map. There is little literature on methods that can natively utilize 2.5D or 3D information and thus use a less compressed representation of the environment for planning. In this work, methods from both groups were systematically compared. A suitable simulator and physics engine have been chosen to enable a realistic evaluation of 2.5D navigation in a simulation. For the methods using the 2D view, classical and widely used planning algorithms were used. To generate the map for the classical methods, a 2.5D map was converted into a 2D map using slope information. The classical search algorithms find a path based on costs on the 2D map. To test a method that uses native 2.5D data for planning, a novel approach was developed that uses the robot's orientations on a 2.5D elevation map. This method samples different locations on the 2.5D map and considers the attitude of the footprint for each position to generate the cost. The evaluation showed that the proposed method, which uses 2.5D data directly, planned shorter and faster paths in most scenarios, while the journey remained safe and reliable for the robot. The results for the classical, 2D methods showed that they are especially useful in scenarios where low computational power is available.**

***Index Terms*— Path planning, Autonomous robots, rapidly-exploring random tree**

I. INTRODUCTION

Path planning for autonomous ground vehicles is usually done on 2D costmap [6] using a search algorithm. The most popular way of using a costmap is projecting objects detected in data from a sensor, like LIDAR or camera, to a cell on the map, where each position is represented either to be occupied or free. More advanced solutions take advantage of the value-range of a 2D costmap in the Robot Operating System (ROS) [11] and map a probabilistic value to each cell, which results in a gradient of values instead of binary, lethal, and non-lethal costs. Both approaches compress the information gathered by a 3D LIDAR or depth camera while losing information on the terrain. While this approach can work well if tuned correctly, it often leads to longer paths, higher energy consumption, and potentially missed opportunities due to the conservative representation of the robot-terrain interaction in 2D. To combat this issue, state-of-the-art solutions use a 2.5D representation of the environment to increase the available information on the map and plan more intelligently by being aware of the actual terrain surface. Planning in 2D also makes it harder to take the physical properties of the robot, such as the maximum tilt angle, into account while planning on off-road areas.

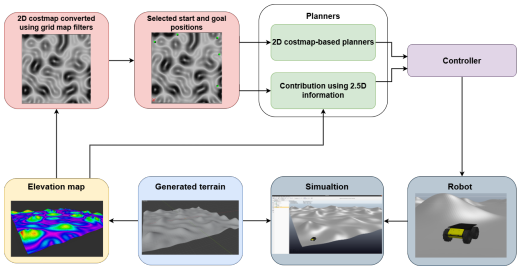

Fig. 1. Flowchart showing the implemented pipeline for testing and evaluation.

The aim of this paper is to compare planning methods in kinematically challenging, accurately simulated off-road environments to gain a better understanding of the properties and limits of path planning algorithms for 2.5D environmental representations. As 2.5D perception goes beyond the scope of this work, it is assumed to be perfect. For the simulation, the widely used robot Husky from Clearpath Robotics [2] will be used, which is a rigid-body off-road differential-drive robot. There are also a lot of resources available to be able to set up a realistic simulation for the Husky.

The contribution of this work can be summarized as follows:

- implementation of a method that compresses 2.5D data to a 2D costmap with minimized loss of information, that can be used by standard search algorithms.
- a method using 2.5D information natively to estimate traversability for complex terrains.
- a simulation pipeline with realistic physics and terrain generation to ensure close-to-life results.
- an evaluation pipeline that is modular, and can be easily adapted to other path planning techniques in the future for further research.

II. RESEARCH

For planning a path, in general, the robotics community has a broad range of well-established methods. Advances in formulating and solving search and optimization problems have finally enabled advanced research in path planning in higher dimensions for drones and off-road vehicles, which would have been impossible previously. Most solutions rely on a well-tuned 2D obstacle map where lethal and non-lethal obstacles are defined to cover the worst-case scenarios and

*This work was funded by the Austrian defense research program FORTE of the Federal Ministry of Finance (BMF) under the project PATH.

[1]{akoczka,gerald.steinbauer-wagner}@tugraz.at, Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Graz, Austria.

https://doi.org/10.34749/3061-0710.2025.15

allow for safe traversal. These solutions are computationally efficient, however, they lack a full understanding of complex environments. However, novel methods exploit 2.5D data natively to allow for more intelligent path planning.

The ArtPlanner [17] by Fankhauser et al. from ANYbotics [1] is one of the few planning pipelines using native 2.5D information for planning. Their approach requires knowledge about the robot's shape and kinematic abilities. They utilize a sampling-based approach with LazyPRM* at its core, which checks the feasibility of a pose at any given sample point. Their approach uses the whole body of the robot to check for collisions and kinematically feasible positions using an elevation map.

Traversability-Based RRT* [16] by Takemura et al. was originally developed for a planetary rover model. The authors approach the problem without a mapping algorithm, planning the path directly on the perceived pointcloud. They use RRT* to sample the points and project the robot's footprint to the terrain, which they use to calculate an orientation-dependent cost. The quality of the path in this case is highly dependent on the LIDAR's update frequency and point density.

"Risk-Aware Mapping and Planning"(RAMP) [14] from Sharma et al. uses a compressed 2D representation of the environment. The authors of the paper approach the problem from the controller's perspective, improving known tools and algorithms to better understand traversability of known and unknown terrain. The problem the authors describe is the lack of awareness of known and unknown spaces in the control phase. The authors use elevation mapping and compress the 2.5D representation into a 2D costmap to make it more efficient and use an improved path planning approach on the 2D representation to solve the described problem.

In the paper "STEP: Stochastic Traversability Evaluation and Planning for Risk-Aware Off-road Navigation" [8] from Fan et al. a complete navigation pipeline is presented. The pipeline aims to solve planning challenges in extreme off-road situations. The authors approach the problem similarly to RAMP by using elevation mapping for the global planning problem and converting the 2.5D representation into a 2D traversability map with a custom cost function. They, however, couple this approach with the control and path following problem as well, including the full 2.5D representation of the environment in the control problem, by calculating a kinematics-based cost for traversal. They approach this similarly to [16] from Takemura et al. , by using the orientation of the robot on a 2.5D map.

III. Evaluation Design

As presented in the previous section, state-of-the-art works in off-road planning take one of two approaches:

- pipelines, that use a combination of mapping techniques and costmap calculations, but plan using standard search algorithms on a 2D costmap
- methods which extend their cost calculations with native 2.5D data, using the kinematic properties of the robot

In order to evaluate if the two approaches are suitable for path planning in challenging off-road environments instances of each approach are prepared and systematically evaluated in a standardized, simulated setting. The first part of the implementation and tests, corresponding to the first bullet point in the above list, was influenced by the works STEP [8], and RAMP [14], using a selection of search algorithms based on the suggestions of the paper [10]. The second part of the evaluation, expanding an algorithm to use 2.5D data natively has been inspired by the techniques used by the ArtPlanner [17] and Traversability-based RRT* [16]. This method has been implemented from scratch, as no openly available solution was available to test at the time of writing. The two distinct approaches are implemented in ROS and simulated using CoppeliaSim [12], which supports both ROS1 and ROS2. We perform the evaluation in simulation first as extensive tests with hardware and real environments are not feasible. It has been determined in previous work, that the best physics engine for simulating rolling and bouncing (both of which are important properties for the Husky robot) is Bullet. This is important for the realistic simulation of robot-terrain interactions. This is also supported by the benchmark of Kang et al. in [5] and Farley et al. in [9]. The work of Farley et al. [9] includes the model of the Husky robot and a corresponding Lua script for CoppeliaSim which will be used as a basis and extended to extract more data for the evaluation.

A. Evaluation metrics

The most problematic property of off-road traversal is the fact that uneven terrain easily pushes the hardware to its limits. This means that finding routes that don't exhaust the robot's capabilities to a dangerous level while keeping the path reasonably short and quickly traversable, while consuming as little energy as possible, are the most important goals of such a path planner. Thus, we developed a set of metrics for our evaluation of path planning algorithms for off road environments:

- **standard deviations of roll and pitch** are expected to be lower for the 2D-based algorithms, as these will try to minimize the cost on the slope-based costmap.
- **the length of the planned path** is estimated from 2D coordinates returned by the algorithms.
- **traveled path length calculated using the odometry** is expected to deviate from the generated path due to slippage and sharp turns.
- **required energy** to complete a path gives a good indication of planning quality. Paths produced by the two methods are expected to deviate in energy consumption.
- **average and standard deviation of torque of the robot joints** indicate the average of the momentary efforts the robot had to make along the path. This metric is expected to be higher for more aggressive plans.
- **travel time from sending the goal to reaching the goal** is expected to be correlated with path length, however, it might deviate due to slippage if a low quality path is provided which is harder to travel along.
- **planning time** is the time from issuing the goal to receiving a path from the planner.

https://doi.org/10.34749/3061-0710.2025.15

- **successful traversal** of the generated path gives an indication if the planner has provided a traversable path. This metric uses a timeout of 6 minutes for reaching the goal.

IV. Methodology and Implementation

In order to evaluate the quality of a path provided by the path planners, a complete system of perception, planning, and execution is needed, as the robot will be executing the plans in the simulation system. The building blocks of this system are described in this section.

A. 2.5D grid map and conversion to 2D

The first method, which follows the style of papers [8] and [14], implements a conversion mechanism, that filters the 2.5D grid map and converts it into a 2D costmap while maintaining terrain awareness. By doing this, we ensure, that the initial source of information (the elvation map) ist the same for both methods, while retaining the maximum amount of information in 2D. The conversion is done using a slope filter, which converts slope data into cost values on the 2D costmap. This method works by applying a mathematical filter to the elevation map, which calculates the normal vector of every cell on the map, and takes the arc-cosine of these normal vectors. This results in a map of slope values in radians. It is known from the datasheet of the Husky, that its maximum claimable slope is 30 degrees. It is known from initial testing, that the realistically climbable slope on a smooth, rocky surface is around 25 degrees. By adjusting the maximum degree of slope to be 25 degrees, any slope that's over this value will be clipped to the maximum lethal value on the map. This results in a gradient of slope values between 0 and 25 degrees, which is a good representation of where the robot can safely traverse. The cost of a cell on the map thus looks as follows:

$$C(x,y) = min(255 \cdot \frac{\alpha(x,y)}{27}) \tag{1}$$

where α represents the calculated slope value in degrees at given map coordinates. The algorithms that will be used for standard planning on a 2D map are **A*, Theta*, D*, and RRT*** which were selected based on the decision tree shown in [10] by Gargano et al. Even though A* would be sufficient to test as a graph-based search algorithm, due to its wide usage and popularity, its newer iterations, D* and Theta* have also been investigated to see if they have any disadvantage for future work. Thus, the algorithms Theta*, and D* are expected to perform similarly or even the same as A*, because the underlying heuristics guarantee an optimal solution for each of these algorithms. Theta* might perform worse than A*, due to it connecting line-of-sight nodes without considering the costs between them. RRT* is considered for its speed, efficiency, and asymptotic optimality. This is also the algorithm, which will be extended to use 2.5D information directly. The cost function of each algorithm, including heuristics, looks as follows:

$$C(n) = g(n) + h(n) + \gamma(n) \tag{2}$$

where $\gamma(n)$ represents the cost of the cell at position n shown in Equation 1. $C(n)$ is then the estimated cost of the path from the start node until the goal, taking node n. The cost $g(n)$ is the already accumulated cost until the last node before expansion, and $h(n)$ is the heuristic cost, which in the case of A* is the Euclidean distance to the goal.

B. Extended RRT*

The second method uses the height and terrain data of the 2.5D grid map natively during planning, similar to the works in [17], [16] and [8] by using the robot's footprint to determine the attitude. The most important factor when planning on rough terrain is the locomotion capabilities of the robot. This is strictly bounded by its kinematic limits, like tipping angle, and max climbing angle. Following the idea of [16] RRT* is extended with an additional function, which projects the robot's footprint onto the 2.5D data and determines a cost from its attitude. Instead of doing this on a pointcloud as shown in the paper [16], it is done on a 2.5D grid map. This approach has the advantage, that a uniformly distributed map is used, regardless of point density, and independent from the capabilities of the sensor. The choice of RRT* ensures that given enough time and samples, the solution should converge to an optimal path. It is however planned for future work to also test other sampling-based methods as a basis for this contribution, like LazyPRM*, that is also used by the ArtPlanner [17]. By projecting the robot's footprint onto the grid at any given point, the representation of the robot's position and attitude can be described with the translation and attitude of its footprint in space. The projected plane (called pseudo-plane in [16]) has a normal vector on the surface of the plane at an arbitrary sampling point, which at any given time is solely dependent on the plane's position relative to the grid map. From this normal vector, the roll and pitch values can be extracted and used in a cost function. The cost function is a weighted sum of the values, similarly to the method shown the paper [16]. The yaw or rotation of the currently sampled point on the plane is dependent on the previous (parent) node in case of RRT*, and is calculated by taking the angle between the parent and currently sampled node in reference to the map's frame. The four wheels of the robot form a rectangle. Projecting a rigid rectangle onto a height map is not a trivial problem. If two opposing corners of a rectangle are at different heights, it leads to a diagonal split in the middle, which separates the rectangle into 2 triangles, which leads to two possible normal vectors, one for each half, without any obvious method to choose one. To simplify this problem, an assumption is made: the two rear wheels of the robot will only be considered at one point, in the middle, forming a triangle of the robot. While this isn't ideal, it is still a good representation of the possible orientation of a rigid-body robot when projected onto a plane. The footprint is illustrated for the Husky robot in Figure 2 on the left. Now, projecting these three points onto a height map will always result in a triangle with an even surface. More importantly, the normal vector can be easily calculated now, by taking the cross product of two

https://doi.org/10.34749/3061-0710.2025.15

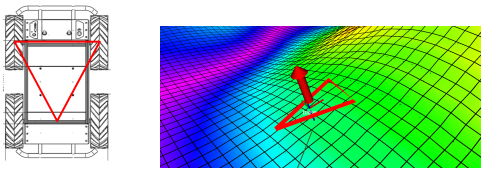

Fig. 2. The triangle formed between the front two wheels and the average of the rear wheels illustrated on the Husky robot on the left, and the normal vector along with the projected triangle footprint visualized in RViz on the 2.5D grid map on the right.

sides of this triangle.

An obvious issue with this method is the fact that if a small, but lethal obstacle falls into the area of the triangle, it won't be considered by the algorithm at all. This wasn't an issue for the tests conducted in work, but this case must be handled in the future for real-life tests. Taking the cross product of two of the edges of the projected triangle, and normalizing it results in the normal vector of the plane. To calculate the pitch and roll of the normal vector, the following function is used:

$$g(k) = \begin{bmatrix} \phi_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} atan2(n_x, n_z) \\ -atan2(n_y, n_z) \end{bmatrix} \tag{3}$$

where n_x n_y and n_z represent the components of the calculated normal vector. The resulting normal vector is visualized in RViz in Figure 2 on the right.

Pitch and roll can then be used to form the following cost function:

$$C(k) = (W_\phi \frac{|\phi_k|}{N_\phi} + W_\theta \frac{|\theta_k|}{N_\theta}) W_s \tag{4}$$

ϕ represents pitch, and θ represents the roll values calculated in the previous step. For simplicity, in this proof-of-concept implementation, only the absolute value of these are taken, however, the option to use the sign of these variables is left for future work, as it might be useful to penalize going up or downhill. The factors N_ϕ and N_θ are normalization values, which have been defined empirically to be 100 and serve the purpose of scaling the values, such that they aren't dominant in comparison to the distance cost. Furthermore, the weight factors W_ϕ and W_θ have been defined to be able to adjust the influence of each factor independently, while W_s adjusts the influence of the entire expression. The weight values have been set up to be adjustable using a feature of ROS called "dynamic reconfigure" [4]. Additionally to the cost-calculation, an initial feasibility-check is done to ensure that the roll and pitch values are within the maximum range, and the sample is thrown away if it isn't the case.

C. Terrain generation and simulation setup

The ground truth terrains are generated using Blender with its built-in geometry node plugin using ridged-multifractal noise [3] which uses Perlin noise[7] at it's basis. The point cloud for elevation mapping is also exported during this process with high density. The main limitations of map size are the file size for storage and the processing power needed to use them. The maps used are as large as it was practically possible, with 50m x 50m. The tested algorithms are assumed to work seamlessly on a real setup, using a much smaller, rolling map, if they are able to achieve good enough performance on such a large map. For this work, a total of 5 random terrains have been generated randomly to represent different terrain difficulties. Moreover, a 6th one was created manually to visualize the benefit of the 2.5D planning method. This will be shown in Section V. The terrains are then imported manually into CoppeliaSim. The starting position of the robot has been chosen to be a random corner of each map. To increase the number of planning problems, the opposite corner is used on each map as a second starting position. This effectively leads to 10 different terrains from the perspective of the planning algorithm. For each terrain, 5 goal positions are distributed on the map. These goal positions have been chosen empirically, by driving the robot manually and making sure that the positions are actually reachable. In future work, this process shall also be automated to be able to test an arbitrary number of terrains and goal positions. 5 terrains with 2 starting positions and 5 goals each lead to 50 unique planning problems for each of the tested algorithms. An example terrain with marked start and goal positions is shown as a 2D costmap in Figure 3. To

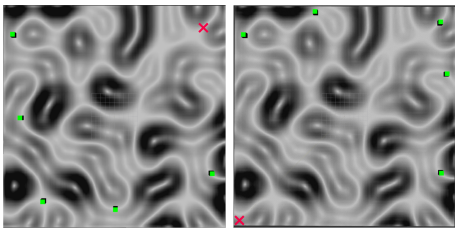

Fig. 3. Example goal positions on a generated terrain. The robot's initial positions are indicated by a red X in the bottom left corner and in the upper right corner.

evaluate energy consumption, the Lua script in CoppeliaSim has been extended to progressively calculate and publish the cumulative energy used by the virtual motors of the Husky robot to a ROS topic.

D. Evaluation

For testing, a Python script has been developed which automates the process, loads the simulation, and saves all the recorded data. For every algorithm, the program iterates through generated terrains, and for each terrain, the corresponding navigation tasks (goal positions) are executed. During execution, the script receives data from ROS messages continuously. All data is saved into .csv data frames and array files, which are evaluated later using another automated script. Saving all raw data also makes it possible to evaluate any run using other criteria in the future. The flow of the automation script is shown in Figure 4. For path execution, a basic version of Timed-Elastic-Bands(TEB) [13] has been used as a controller (without object avoidance), and tuned empirically to follow the generated paths as strictly as possible, with as little influence as possible.

https://doi.org/10.34749/3061-0710.2025.15

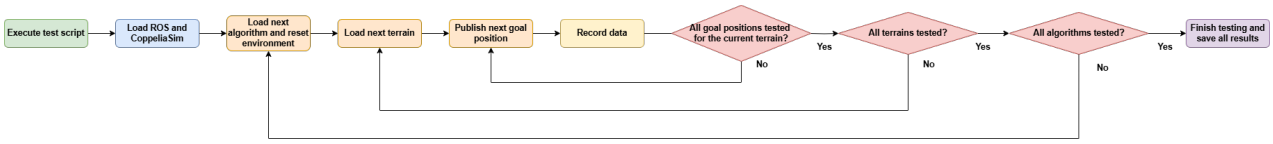

Fig. 4. The flowchart showing the process of the evaluation, executed by the test script. The script is started manually and iterates through a list of registered algorithms, terrains and goal positions, which can be of arbitrary length.

V. Results

The results will show averaged metrics over all planning problems. The final results have only been calculated for runs that all algorithms completed successfully so that the comparison stays fair (union of all data where execution succeeded). A successful execution was defined by reaching the goal within 6 minutes after generating a path.

First, the number of failed executions will be shown (see Figure 5). Please note that for RRT* and the proposed, extended RRT*, which will be called "RRT* Kin." (kinematic) in the plots, the worst-case number of failures of 5 full sets of navigation tasks is given. The most frequent cause of a failed

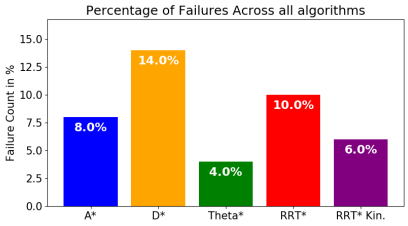

Fig. 5. The percentage of total failures across algorithms. RRT* and RRT* kinematic shows the worst case failures of all the runs.

execution was turning on slopes, where a differential drive robot struggles the most, as it loses friction while also having a shifted weight distribution. The proposed method has only failed due to this issue. Other algorithms have frequent cases of getting stuck in tight corridors or valleys due to the lack of terrain awareness.

Perhaps the most important finding of all is the average effort in watt-hours (see Figure 6). It was expected that the proposed extended method, RRT* kinematic, would produce paths that consume more energy because it takes more aggressive paths. However, due to the fact that on average it produced 5% shorter paths than the next best algorithm while having a low variance in yaw (less turning) made it the best regarding power consumption in the tested scenarios.

By taking shortcuts at safe places, which are within the robot's kinematic capabilities, the proposed method was less susceptible to turning on slopes and also made less sharp turns leading to an overall shorter path. This behavior is also confirmed by the results shown in Figure 7, which shows the cumulative height difference.

Looking at Figure 7 it can be concluded, that the higher cumulative height difference did not have a negative influence on travel times, making the proposed method the fastest by a small margin. It also shows that travel times are more

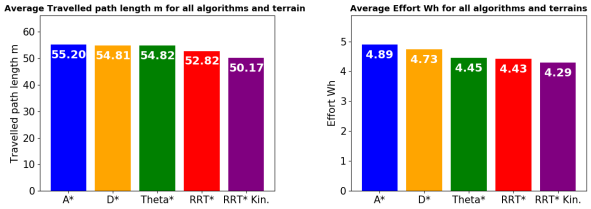

Fig. 6. The averaged traveled path lengths in meter on the left, and the averaged energy needed (effort) in Wh across all algorithms, terrains, and paths.

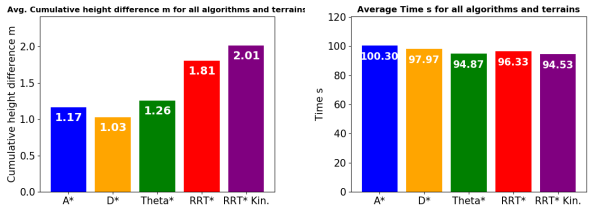

Fig. 7. The cumulative height difference of traveled paths on the left and the mean of travel time on the right, averaged over all paths and terrains.

correlated with the path length than with the cumulative height, which means that all algorithms managed to find more-or-less easily traversable trajectories for the robot.

Looking at the mean standard deviation of roll and pitch shown in Figure 8 together with traversal time in Figure 7 suggests that taking a smoother path only has a small negative impact on travel time. Thus, in some cases it might be more feasible to take a flatter path. It was expected that the standard 2D costmap-based algorithms would produce a path with fewer variations as they only use a limited amount of information, and prefer lower-cost cells.

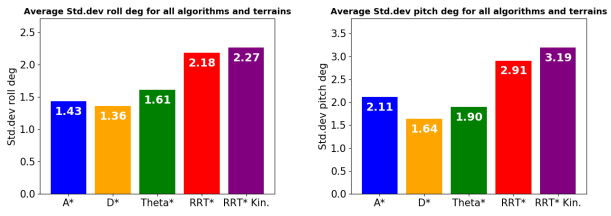

Fig. 8. The averaged standard deviation roll and pitch across all algorithms, terrains, and paths.

Unexpectedly, the vanilla RRT* algorithm produced a high cumulative height difference, while also traveling longer.

https://doi.org/10.34749/3061-0710.2025.15

This could be explained by the fact that it produces similar paths to Theta*, with lots of straight sections, but having much more of these sections, leading to more unwanted zig-zag turns. Figure 6 showed that the proposed method actually consumed about 3% less energy than the next best algorithm. Taking the standard deviation of yaw, pitch, and roll, and the cumulative height into account, it can be concluded that it's most of the time more efficient to go above a hill than to go around it if it can be guaranteed that the robot can operate within its limits.

Until now, the results are promising, especially due to the fact that the proof-of-concept method already performs well in the initial version. Unfortunately, a significant disadvantage of the proposed method is planning time. On average, the proposed solution returns a path in 7630ms for the given map size, which is very high in comparison to the 30-300ms path-return times of the classical 2D search algorithms. However, looking at the unmodified RRT*, the planning time only decreases by about 900ms to 6725ms. This shows that the added cost calculations using the robot's footprint have a relatively low impact on the planning time. Thus, the implementation has been determined to be inefficient, and as a next step, the solution will be re-implemented using the popular Open Motion Planning (OMPL) [15] library to improve speed.

In order to visualize the advantage of the proposed method even better, a map with a single ramp with a slope of 29 degrees has been made. This is above the set worst- case cost factor of the classical algorithms, and thus all of them fail to provide a path to the goal. However, the proposed solution can find a skewed path that stays within the bounds of the robot, as the slope is less steep from a skewed perspective. This is shown in Figure 9.

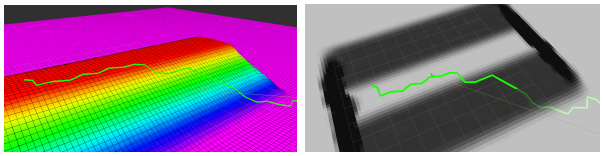

Fig. 9. Planning skewed to the slope makes the slope less steep.

VI. SUMMARY AND OUTLOOK

The evaluations provided us with valuable information regarding planning on uneven terrain. It has been shown that using 2.5D information natively is beneficial for planning in off-road scenarios by evaluating different metrics recorded during testing. It has also been concluded, that the proposed approach is less computationally efficient than methods using standard search-based algorithms on a 2D costmap. Thus, there is still work to be done for the developed 2.5D-based planner to be usable in real-time with a real robot. For the 2D approach one of the most basic and widely used methods has been used. It compresses the terrain data into a slope map, which results in the least amount of loss of information about the environment. In reality, the slope map could be extended by additional traversability costs to also account for different surface types and roughness levels. Meanwhile, the proposed method using 2.5D data directly has great potential of combining the cost calculation on a 2.5D surface with other types of map layers to make planning more efficient. As the proposed method is only a proof-of-concept solution, it still requires more work to achieve a better quality outcome. Path executions failed in most cases due to turning on slopes. In future work, a converted slope map could be taken into account by the controller to restrict turning on steep slopes, similar to the paper [14], which would mitigate this issue.

REFERENCES

[1] "ANYbotics — Autonomous Legged Robots for Industrial Inspection — anybotics.com," https://www.anybotics.com/, [Accessed 23-12-2024].

[2] "Clearpath Robotics: Mobile Robots for Research & Development — clearpathrobotics.com," https://clearpathrobotics.com/, [Accessed 02-01-2025].

[3] Noise Texture Node - Blender 4.3 Manual — docs.blender.org. https://docs.blender.org/manual/en/latest/render/shader_nodes /textures/noise.html. [Accessed 05-01-2025].

[4] M. C. Blaise Gassend, "dynamic reconfigure - ROS Wiki — wiki.ros.org," http://wiki.ros.org/dynamic_reconfigure, [Accessed 11-01-2025].

[5] J. H. Dongho Kang, "SimBenchmark — leggedrobotics.github.io," https://leggedrobotics.github.io/SimBenchmark/, [Accessed 30-12-2024].

[6] D. H. Eitan Marder-Eppstein, David V. Lu, "costmap_2d - package summary," http://wiki.ros.org/action/info/costmap_2d?action=info, 2018, [Accessed 22-12-2024].

[7] T. R. Etherington, "Perlin noise as a hierarchical neutral landscape model," *Web Ecol., 22, 1–6*, 2022.

[8] D. D. Fan, K. Otsu, Y. Kubo, A. Dixit, J. Burdick, and A.-A. Agha-Mohammadi, "Step: Stochastic traversability evaluation and planning for risk-aware off-road navigation," 2021. [Online]. Available: https://arxiv.org/abs/2103.02828

[9] A. Farley, J. Wang, and J. A. Marshall, "How to pick a mobile robot simulator: A quantitative comparison of coppeliasim, gazebo, morse and webots with a focus on accuracy of motion," *Simulation Modelling Practice and Theory*, vol. 120, p. 102629, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1569190X22001046

[10] I. E. Gargano, K. D. von Ellenrieder, and M. Vivolo, "A survey of trajectory planning algorithms for off-road uncrewed ground vehicles," in *Modelling and Simulation for Autonomous Systems*, J. Mazal, A. Fagiolini, P. Vasik, F. Pacillo, A. Bruzzone, S. Pickl, and P. Stodola, Eds. Cham: Springer Nature Switzerland, 2025, pp. 120–148.

[11] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," vol. 3, 01 2009.

[12] E. Rohmer, S. P. N. Singh, and M. Freese, "Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013, "www.coppeliarobotics.com".

[13] C. Rösmann, F. Hoffmann, and T. Bertram, "Timed-elastic-bands for time-optimal point-to-point nonlinear model predictive control," in *2015 European Control Conference (ECC)*, 2015, pp. 3352–3357.

[14] L. Sharma, M. Everett, D. Lee, X. Cai, P. Osteen, and J. P. How, "Ramp: A risk-aware mapping and planning pipeline for fast off-road ground robot navigation," 2023. [Online]. Available: https://arxiv.org/abs/2210.06605

[15] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.

[16] R. Takemura and G. Ishigami, "Traversability-based rrt * for planetary rover path planning in rough terrain with lidar point cloud data," *Journal of Robotics and Mechatronics*, vol. 29, pp. 838–846, 10 2017.

[17] L. Wellhausen and M. Hutter, "Artplanner: Robust legged robot navigation in the field," *Field Robotics*, vol. 3, no. 1, p. 413–434, Jan. 2023. [Online]. Available: http://dx.doi.org/10.55417/fr.2023013

https://doi.org/10.34749/3061-0710.2025.15