GPU-Accelerated 3D-Aware Model Predictive Path Integral Control for High-Speed Autonomous Lunar Navigation

by

Nathan Nascimento

to obtain the degree of *Master of Science*
at the *Delft University of Technology*,

to be defended publicly on *Wednesday July 2nd, 2025 at 14:30*.

Student number: 6072305

Thesis project duration: December, 2024 - June, 2025

Thesis committee:

Dr. Javier Alonso-Mora	TU Delft, Academic supervisor
Dr. Laura Ferranti	TU Delft
Dr. Miguel Olivares Mendez	SpaceRobotics Group, University of Luxembourg
Dr. Antoine Richard	NVIDIA, Daily supervisor

GPU-Accelerated 3D-Aware Model Predictive Path Integral Control for High-Speed Autonomous Lunar Navigation

***Abstract*— In the context of lunar exploration missions, autonomous navigation is essential to limit reliance on human intervention, which is hindered by significant communication delays and limited bandwidth. However, due to constrained onboard computational resources, complex robot-soil interactions, and the inherent challenges of full autonomy, most lunar rovers are currently limited to speeds of around 10cm/s. In this work, we investigate the use of Model Predictive Path Integral Control (MPPI) for high-speed navigation on uneven and cluttered lunar terrain. To this end, we develop an MPPI controller using the WARP framework, allowing to easily leverage the inherent parallelisation capabilities of GPUs. Experiments were conducted with a simulated Husky rover navigating realistic lunar terrain generated from NASA data, demonstrating the effectiveness of our motion planner. Since most MPPI implementations assume flat ground when sampling trajectories, this simplification can introduce inaccuracies, resulting in suboptimal, and sometimes even risky paths. To address this limitation, we propose a lightweight mathematical method for 3D trajectory projection, and evaluate its performance against the standard 2D MPPI approach. Overall, simulation results show that our 3D-enhanced MPPI significantly outperforms its 2D counterpart in terms of both traversal speed and obstacle avoidance on sloped, rock-scattered terrain. On flat ground, however, the traditional 2D MPPI still exhibits a high level of robustness. These findings suggest that an adaptive strategy, dynamically switching between 2D and 3D trajectory projection based on local terrain features, could offer a valuable trade-off between performance and power efficiency.**

I. INTRODUCTION

The long distance between Earth and the Moon causes important communication delays that make teleoperation suboptimal and less reliable for controlling rovers [1]. It also comes with severe bandwidth restrictions, limiting the amount of data that can be sent in real-time to an Earth station. Hence, the development of autonomous navigation is crucial to enable robots to explore complex terrain without constant human supervision. Reaching autonomy in such an environment can be highly challenging. On the one hand, rovers have limited onboard processing capabilities and energy resources. On the other, the terrain topography presents many constraints including slopes, loose regolith, craters and rocks of varying sizes, as shown in Fig. 1. The optimisation of lunar navigation requires a robust motion planner to not only compute viable paths but also determine the appropriate kinematic behaviour.

As of today, most lunar rovers operate typically at an average speed limited to around 10cm/s [2]. If the life-span of lunar rovers was long, this would not be an issue. However, most modern rovers are only rated to survive a couple of weeks at best on the lunar surface. Together with the low speed of the rovers, this strongly limits the range and scope of the robotics lunar missions. This limitation is due in part to computational constraints, as motion planning requires real-time processing and control. Achieving higher speeds up to 2m/s would enable rovers to complete more missions within their limited lifespan. NASA Ames, for instance, recently developed VIPER [3], a rapid lunar rover that explores the Moon's surface looking for ice or other specific resources, travelling at speeds up to 0.2m/s.

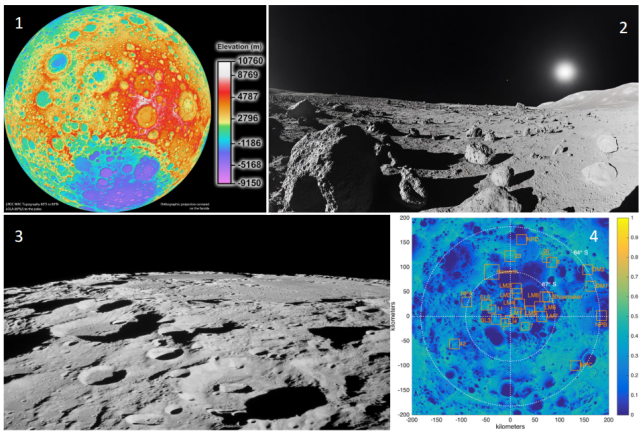

Fig. 1: (1) Topographic map of the Moon. (2) Lunar terrain scattered with rocks of varying sizes. (3) Lunar terrain with high crater density. (4) Average sun exposure map.

Although the existing literature includes effective motion planners for lunar applications [4, 5, 6], there is still interest in exploring alternative approaches. Specifically, most current lunar planners were not originally designed to support high-speed traversals. On Earth, recent research has demonstrated the potential of sampling-based Model Predictive Control (MPC) techniques for aggressive navigation where several objectives must be jointly optimised [7, 8]. These models also accommodate non-differentiable kinematic models, allowing for greater flexibility. Thus, it seemed interesting to investigate the use of such controllers to increase the average speeds during long-distance traverses on the Moon. In this study, we focus on the use of a sampling-based MPC method known as Model Predictive Path Integral Control (MPPI) [9]. This motion planner computes control inputs by evaluating multiple stochastically

sampled sequences of commands to minimise a cost function. As lunar navigation introduces a combination of multiple constraints and objective functions, the computational burden of MPPI controllers can become overwhelming. To address this issue, prior works have highlighted how well-suited this type of controller is for parallel calculations[10], making it ideal for GPU acceleration. And while GPUs could have been considered too power hungry a couple years back, modern days SOC like the ones in Nvidia's Jetson boards offer a good trade-off between performance and power consumption.

Another key limitation of most existing motion planning approaches, including MPPI, is their reliance on a flat-ground assumption when predicting future trajectories [11]. Although this simplification can be computationally efficient, it fails to account for the three-dimensional nature of the lunar terrain. In particular, yaw variations induced by sloped surfaces and especially craters can cause a rover to deviate significantly from a planned 2D trajectory. This can lead to navigational errors, increased energy consumption, and potentially cause critical events such as rollovers or collisions. On the moon, there are no second chances, and these events must be avoided at all costs.

Two contributions are presented in this thesis. First, we propose an implementation of MPPI using WARP [12], a python-based framework allowing for JIT compilation of GPU kernels enabling users to write highly parallelised code. This controller was successfully evaluated in Nvidia's OmniLRS environment in IsaacSim [13], where a Husky rover navigates through large-scale, realistic lunar terrains generated from high-resolution digital elevation maps (DEMs) provided by NASA. Secondly, we introduce a trajectory prediction model that accounts for the 3D surface. The mathematical model chosen to perform the projection was purposely kept as simple as possible to keep the algorithm as computationally light as possible. The impact of using the 3D trajectory prediction method instead of 2D is evaluated by comparing the MPPI performances using both methods.

This thesis is structured as follows: Section II reviews existing literature on GPU-accelerated MPPI implementations and various trajectory planning methods for uneven terrain, with a particular focus on approaches that integrate 3D trajectory prediction into the MPPI controller. Section III introduces the fundamental concepts necessary to understand the core of this thesis, including an overview of classic MPPI controllers, as well as the general principles of GPU computing, and the planning approach specific to lunar navigation. In Section IV, the mathematical approach for 3D trajectory planning is detailed, followed by the formulation of the cost function and the low-level controller used alongside MPPI. Section V describes the implementation of the MPPI controller on GPU and details how the navigation pipeline manages interactions between the CPU and GPU. In Section VI, the experimental setups and evaluation metrics used to assess the contributions of this thesis are described. Section VII presents the outcomes of these experiments, accompanied by relevant illustrations. The findings are further discussed, along with the limitations, in Section VIII. Finally, Section IX summarises the conclusions drawn from this research.

II. RELATED WORK

Numerous motion planning algorithms have been investigated for autonomous navigation, though their suitability for lunar environments varies significantly. Reactive manoeuvre methods [14], such as the Dynamic Window Approach (DWA) [15], the Artificial Potential Field (APF) method [16], and Velocity Obstacles [17], rely on binary occupancy grids and are therefore limited in their ability to handle environmental constraints like terrain topography, energy consumption or illumination and temperature [18]. Some variants like the Energy Efficient DWA [19] or extended DWA [20] were introduced to account for the energy consumption. However, those are only based on the commanded speed, and still do not account for the surrounding environment properties. The APF method framework can possibly be extended to incorporate additional terrain features such as slope. Yet, it is prone to getting stuck in local minima [21], and scales badly in environments with a large number of constraints [22].

By comparison, horizon-based methods like MPC aim at finding an optimal trajectory over a finite prediction horizon, considering constraints and using robot dynamics [23]. Those methods have demonstrated strong performance in motion planning across different applications [24, 25, 26]. One of their drawbacks is that they remain particularly computationally intensive [27, 28], especially with resource-limited rovers. Another limitation of optimisation-based MPC is its reliance on differentiable cost functions and dynamic models, which is even more pronounced in off-road conditions due to complex wheel-terrain interactions [29].

To address this, sampling-based variants such as MPPI control have emerged, which evaluate a distribution of sampled control sequences to approximate the optimal policy [8]. MPPI has already proven effective in off-road scenarios and in high-speed applications [30, 31, 32, 33, 7]. Since MPPI's performance strongly depends on sampling density, its computational cost can quickly grow with the number of objectives to optimise. To address this, the potential of MPPI for parallelisation on GPUs has already been highlighted [34].

Based on current knowledge, the only publicly available GPU-accelerated MPPI implementation is MPPI-Generic [34]. It is a C++/CUDA library offering real-time performance and modularity across dynamics models, cost functions, and controller variants. However, this pure CUDA code can be complicated to modify and prototype from. Hence, we looked into building our own WARP based controller using NVIDIA's WARP framework [12]. This framework retains python flexibility while allowing to write compiled CUDA code with ease.

Many existing approaches model vehicle kinematics on rough terrain by assuming low-speed conditions. For instance, Datar et al. [35], Fan et al. [36], and Krüsi et al. [37] rely on simplified kinematic models suited for slow

traversal. However, as speed increases, these models quickly lose accuracy due to unmodeled dynamic effects. To address this, some researchers developed more complex dynamics models capable of explicitly predicting wheel forces and vehicle motion. Yu et al. [38] implemented a complex 3D rigid-body model with tire forces integrated into MPC. While physically accurate, the control rate was limited to 3 Hz, which is more than ten times slower than what is targeted in this study in order to enable safe high-speed traversals. In contrast, Han et al. [7] proposed a different approach combining simple geometry-based methods with inertial dynamics model. Their model was integrated into an MPPI controller and validated in real-world off-road scenarios. However, their method was used to evaluate the dynamic stability of the trajectories, not to predict them. For prediction, they first sampled trajectories on a 2D plane, then orthogonally projected them onto the elevation map. This simplification overlooks how the topography itself can influence the robot's actual trajectory when control inputs are applied, which is the focus of this research.

Existing research has also focused on learning-based approaches to navigate uneven terrain. César-Tondreau et al. [39] applied imitation learning for negative obstacle traversal at speeds around 1 m/s. Although promising during training, performance dropped by over 50% when tested in different terrain geometries, highlighting poor generalisation. Similarly, Gibson et al. [40] and Xiao et al. [41] developed dynamics learning models trained on driving data. A notable limitation of those methods is the collection of data. When going high-speed, collecting data near the safety limits of the vehicle can indeed be dangerous, and of course costly and challenging when considering lunar navigation data. Lee et al. [42] further extended this concept by predicting 6-DOF motion using terrain-aware neural networks and integrating them into an MPPI controller. While this method showed reduced failure rates in simulation, it lacked yaw prediction capabilities. Additionally, it struggled to generalise to irregular terrain, partly due to the smooth and differentiable ground manifolds used in training.

An alternative approach was proposed by Pezzato et al. [43], who used GPU-accelerated simulation in IsaacGym to sample and evaluate 3D trajectories without explicitly modeling dynamics. Although this method offers accuracy, it involves a significant computational burden, limiting its feasibility for real-time use on energy-constrained hardware like rovers.

In summary, most existing approaches either ignore terrain topography in trajectory prediction or incorporate it using methods that are too computationally expensive or not suited for high-speed applications. Additionally, they rarely compare 3D and 2D trajectories to identify when 3D projection is beneficial. In this work, we explore when and how a simple 3D projection model can improve the performance of traditional MPPI-style controllers.

III. BACKGROUND

This section provides the necessary background to understand the remainder of the thesis. An overview of the MPPI algorithm is first given, followed by an overview of the concept of GPU parallelisation, and finally a description of the planning approach chosen for the lunar application.

A. MPPI Overview

Consider a general nonlinear system governed by discrete-time dynamics and a state-cost function of an inputs sequence defined as follows [9]:

$$x_{t+1} = A(x_t, v_t) \quad \text{where} \quad v_t \sim \mathcal{N}(u_t, \Sigma), \tag{1}$$

$$S(X, V) = \sum_{t=0}^{T-1} \phi(x_t, v_t), \tag{2}$$

where $x_t \in \mathbb{R}^n$ is the state, A is the nonlinear dynamics model, $u_t \in \mathbb{R}^m$ is the mean control input, and v_t is the noisy input drawn from a Gaussian distribution centred in u_t with a covariance Σ. The time horizon is T, $X = [x_0, x_1, \ldots, x_{T-1}]$ is the state trajectory, $V = [v_0, v_1, \ldots, v_{T-1}]$ is the sequence of inputs drawn from the mean control sequence $U = [u_0, u_1, \ldots, u_{T-1}]$. The cost function is denoted as ϕ. For compactness, the state-cost function will simply be referred to as $S(V)$.

The core idea behind MPPI is to compute the next T control inputs to apply to the robot to follow a trajectory that optimises the cost function ϕ. The work of Williams et al. [9] was among the first to provide the theoretical foundations of MPPI. To grasp the principles of the algorithm, it is essential to introduce certain mathematical concepts, starting with **free energy**. Free energy quantifies a control system's performance relative to the cost associated with a specific inputs sequence. Letting $\mathbb{G}$ be some probability density over inputs sequences, the free energy F is calculated as:

$$F = \mathbb{E}_{\mathbb{G}} \left[e^{-\frac{1}{\lambda} S(V)} \right] \tag{3}$$

where $\lambda > 0$ is a temperature-like parameter, and $\mathbb{E}_{\mathbb{G}}$ is the expectation over trajectories sampled from $\mathbb{G}$.

The second key concept is **relative entropy**, often referred to as KL-divergence. It measures how different two probability distributions are. Letting $\mathbb{H}$ be another probability distribution, and the respective corresponding density functions being $g(V)$ and $h(V)$, the KL-divergence between them is given by:

$$D_{\mathrm{KL}}(\mathbb{G} \| \mathbb{H}) = \mathbb{E}_{\mathbb{G}} \left[\log(\frac{g(V)}{h(V)}) \right] \tag{4}$$

Given the sequence of inputs V and mean inputs U defined earlier, it is possible to define the distributions of $\mathbb{Q}$ and $\mathbb{P}$, which correspond to the controlled and uncontrolled measures. They are characterised by their density functions $q(V)$ and $p(V)$. The first one is given by:

$$q(V) = \prod_{t=0}^{T-1} Z^{-1} \exp\left(-\frac{1}{2}(\mathbf{v}_t - \mathbf{u}_t)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{v}_t - \mathbf{u}_t)\right) \quad (5)$$

where $Z = ((2\pi)^m|\Sigma|)^{1/2}$. The uncontrolled density function p(V) is defined as:

$$p(V) = q(V|\tilde{U}, \Sigma) \quad (6)$$

where $\tilde{U}$ is some nominal control input applied to the system, and is usually equal to zero, giving:

$$p(V) = \prod_{t=0}^{T-1} Z^{-1} \exp\left(-\frac{1}{2}\mathbf{v}_t^\top \boldsymbol{\Sigma}^{-1}\mathbf{v}_t\right) \quad (7)$$

Using the definition of the free energy of a control system and the KL-divergence, it is possible to show that there is always a lower bound on the cost for any optimal control problem. This implies the existence of an optimal control distribution $\mathbb{Q}^*$, from which the drawn samples should achieve a lower cost than any other control distribution. It was shown in [44] that this optimal distribution can be defined as follows:

$$q^*(V) = \frac{1}{\eta} \exp\left(-\frac{1}{\lambda}S(V)\right) p(V) \quad (8)$$

where η is a normalisation factor.

From there, the objective then becomes aligning the arbitrary control distribution $\mathbb{Q}$ with the optimal one $\mathbb{Q}^*$ by minimising the KL-divergence between them, as illustrated in Fig. 2.

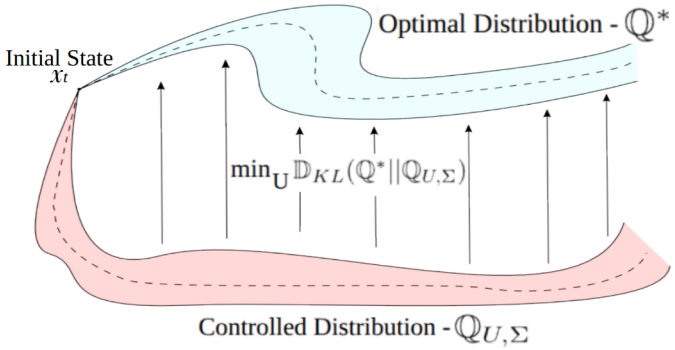

Fig. 2: Process of aligning $\mathbb{Q}$ with $\mathbb{Q}^*$ [9]

Minimising the KL-divergence yields to the following expression for the optimal solution:

$$u_t^* = \mathbb{E}_{\mathbb{Q}^*}[\mathbf{v}_t] = \int q^*(V)\,\mathbf{v}_t\,\mathrm{d}V \quad (9)$$

In order to solve this, it would be necessary to draw samples directly over $\mathbb{Q}^*$, which is impossible since this is the distribution that we aim to estimate. The alternative method suggested is to use the concept of importance sampling:

$$\begin{aligned} u_t^* &= \int w(V) q(V)\,\mathbf{v}_t\,\mathrm{d}V \\ &= \mathbb{E}_{\mathbb{Q}}[w(V)\mathbf{v}_t] \end{aligned} \quad (10)$$

where $w(V)$ is:

$$w_k = \frac{1}{\eta} \exp\left(-\frac{1}{\lambda}\left(S(V) + \frac{\lambda}{2}\sum_{t=0}^{T-1}(2v_t - u_t)^T \Sigma^{-1} u_t\right)\right) \quad (11)$$

Based on the expressions derived above, it is now possible to formulate an algorithm that approximates the optimal control sequence through sampling. The process begins by selecting a time horizon T, which determines the number of discrete time steps, each of fixed duration dt. Then, K sequences of control inputs are sampled from a Gaussian distribution centred on the previous control sequence. Each sampled sequence is then propagated through the system dynamics to generate a corresponding state trajectory. These trajectories are assigned a weight using Equation 11. The approximation of the optimal control sequence U^* is then computed by taking the weighted average of the sampled sequences, as expressed in Equation 10. Finally, the first input of U^* is sent to the robot, and the rest of the sequence is shifted to be used as the next initial control sequence. Fig. 3 exhibits a visual illustration of the MPPI process, while Algorithm 1 provides the pseudocode representation of the MPPI control procedure.

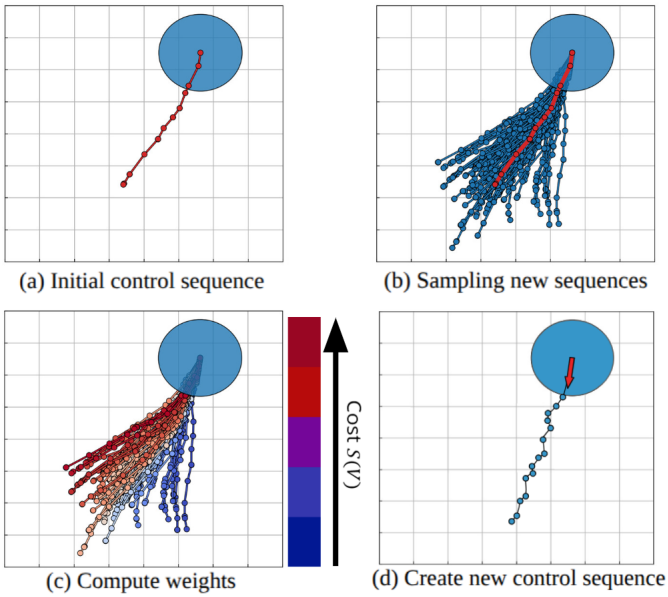

Fig. 3: MPPI process illustration [45]

B. GPU Parallelisation Overview

GPUs are hardware devices designed to execute multiple processes in parallel, making them well-suited for computational tasks such as MPPI, which can handle up to thousands of trajectories per control loop. Unlike CPUs, which are limited to running only a few threads in parallel, GPUs can launch thousands simultaneously. These threads are grouped into units of 32 called *warps*, which are then grouped into blocks and further into grids. Threads within the same block can quickly share data using a *shared memory*, which is both

Algorithm 1 Model Predictive Path Integral Control

Given: A, g: Dynamics and clamping function
K, T: Number of samples and timesteps
dt: Timestep duration
u: Initial control sequence
S: Array of the cost of each trajectory
Σ, λ: Covariance, temperature parameters
ϕ: Cost function
while Goal not reached **do**
  $x \leftarrow$ GetState()
  for $k = 0$ to $K - 1$ **do**
    $S^k \leftarrow 0$
    for $t = 0$ to $T - 1$ **do**
      $v_t^k \leftarrow g(u_t + \epsilon_t^k)$ where $\epsilon_t^k \sim \mathcal{N}(0, \Sigma)$
      $x_t^k \leftarrow A(x_{t-1}^k, g(v_t^k), dt)$
      $S^k \leftarrow S^k + \phi(x_t^k) + \frac{\lambda}{2}(2v_t^k - u_t^k)\Sigma^{-1}u_t^k$
    end for
  end for
  $\rho \leftarrow \min(S^0, S^1, \ldots, S^{K-1})$
  $\eta \leftarrow \sum_{k=0}^{K-1} \exp(-\frac{1}{\lambda}(S^k - \rho))$
  for $k = 0$ to $K - 1$ **do**
    $\omega^k \leftarrow \frac{1}{\eta} \exp(-\frac{1}{\lambda}(S^k - \rho))$
  end for
  for $t = 0$ to $T - 1$ **do**
    $u_t \leftarrow u_t + \sum_{k=0}^{K-1} \omega^k \epsilon_t^k$
  end for
  SendToActuators(u_0)
  for $t = 1$ to $T - 1$ **do**
    $u_{t-1} \leftarrow u_t$
  end for
end while

high-bandwidth and low-latency. This architecture allows for efficient data exchange and coordination between threads.

GPU programs are written as kernels that can be launched with a specified number of threads. In the case of MPPI, the number of trajectories typically defines the kernel's dimensionality. In other words, this means that each trajectory is handled in parallel, enabling to increase the sampling of trajectories with only a minimal impact on the computation time.

C. Planning on the Moon

When navigating on the Moon, motion planners must address physical constraints arising from the lunar topography. The surface is scattered with rocks and boulders ranging in size from small fragments to large obstacles, particularly concentrated around impact craters due to material ejection, as shown in Fig. 4. Additionally, the terrain is uneven, with frequent slopes of varying steepness. These slopes have a direct impact on energy consumption and pose risks such as slippage or instability.

When performing autonomous navigation, motion planning is usually preceded by global planning. This process consists in using the data of the environment to compute a path that connects a start point to a destination while

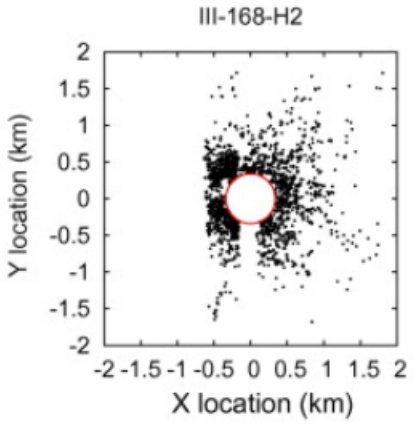

Fig. 4: Clusters of small rocks around a crater [46]

optimising an objective function. This global path is usually converted into a series of waypoints, which are then provided to the motion planner. One of the constraints for traditional motion planners is to adhere to this reference path. Since the robot perceives the local environment in greater detail than the global map allows, the obstacles and constraints that were not represented in the global map can then be taken into account to adjust the path in real-time.

In the lunar context, this process faces two key challenges. First, the resolution of lunar maps is generally low [47], which makes it difficult to capture local features such as small rocks, ground roughness, and craters. Second, some environmental factors like sunlight exposure or temperature that need to be taken into account in the trajectory processing might vary over time. For those reasons, it is complicated, if not impossible, to consider all the constraints during the computation of the global plan.

Facing this challenge, there is interest in exploring a different approach. Instead of generating a detailed global path, the global mapping process produces distanced waypoints that account for larger-scale geometric features. The motion planner will then compute the path as the rover goes between these distanced waypoints, progressively adapting to the local environment. In other words, the MPPI controller will be responsible for the local trajectory planning, while also ensuring to get close to the goal without a predetermined global path.

IV. METHOD

Since the flat-ground assumption does not, by definition, apply to uneven terrain, the trajectory prediction of traditional MPPI may be inaccurate. Instead, the proposed method aims to approximate the robot's motion over 3D terrain using a high-resolution DEM, defined as having a maximum cell width of 10cm. The first subsection focuses on describing the mathematical model developed to predict these rollouts in 3D, taking the terrain's topography into account.

Next, as the target application is lunar navigation, we provide a detailed mathematical formulation of the cost function constructed for this specific context. This cost function will

later be used to assess the impact of our projection method on MPPI performance in challenging lunar terrain. It will also serve to evaluate the effectiveness of the new GPU-based MPPI implementation.

A. Rollouts Projection in 3D

The first step is to convert the inputs into the corresponding linear and angular velocities. To do this, the kinematics model of the robot is needed:

$$\begin{pmatrix} \dot{v} \\ \dot{\omega} \end{pmatrix} = A \begin{pmatrix} v \\ \omega \end{pmatrix} + B \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \tag{12}$$

where v and $\dot{v}$ are respectively the linear velocity and linear acceleration, ω and $\dot{\omega}$ are the angular velocity and angular acceleration, A is the state matrix, B is the input matrix, and $\begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$ is the input vector, corresponding to the left wheel and right wheel inputs, respectively.

Since the traditional MPPI uses the flat-ground assumption, it can use the following basic vector calculus to iteratively compute the trajectory waypoints:

$$\begin{aligned} \vec{d}_{2D} &= \vec{t}_{2D} \cdot v \cdot dt \\ x_{new} = x + d_x, \quad y_{new} &= y + d_y \end{aligned} \tag{13}$$

where $\vec{d}_{2D}$ is the 2D vector which contains the x and y distances travelled, respectively d_x and d_y, during one iteration of dt seconds, $\vec{t}_{2D}$ is the 2D unit heading vector, and v is the linear velocity at this timestep. Given the current heading vector (t_x, t_y), the updated heading vector $(t_{x,new}, t_{y,new})$ is obtained using the angular velocity:

$$\begin{aligned} \theta &= \omega dt \\ t_{x,new} &= t_x \cos\theta - t_y \sin\theta \\ t_{y,new} &= t_x \sin\theta + t_y \cos\theta \end{aligned} \tag{14}$$

Now, as explained, the goal of the proposed model is to account for the terrain elevation in order to compute the trajectories in 3D. The general idea is to perform the 2D propagation that was just described, but within the tangent plane of the current position. That tangent plane must therefore be updated at each timestep to accurately reflect the terrain beneath the robot. This model relies on the assumptions that all four wheels are permanently in contact with the ground, and all belong to the same DEM cell.

The first step is to compute the linear displacement during one timestep. Since the heading vector lies within the current tangent plane, the robot position (x, y, z) is shifted along this heading direction by a distance based on the linear velocity:

$$\begin{aligned} \vec{d}_{3D} &= \vec{t}_{3D} \cdot v \cdot dt \\ x_{new} = x + d_x, \quad y_{new} &= y + d_y \end{aligned} \tag{15}$$

where $\vec{d}_{3D}$ is the 3D vector which contains the x,y and z distances travelled, $\vec{t}_{3D}$ is the 3D unit heading vector indicating where the robot is pointing at. Note that only the x and y positions are updated at this step. Indeed, since the displacement is computed in the tangent plane where the robot is driving, the robot will immediately detach from the surface once it gets out of its DEM cell. To ensure the robot stays in contact with the surface, the z-coordinate is thus updated by using the DEM cell height at the new x and y coordinates. The z-coordinate is obtained by taking the average of the cell's corners heights.

In addition to the z-coordinate, the heading vector must also be updated to remain tangent to the surface. To do that, the equation of the tangent plane at the new robot's position must first be computed. This is achieved by identifying the DEM cell in which the robot is located. Each cell, defined by four corners, can be divided into two triangles. By knowing the height at each corner, it is possible to calculate the normal vector for each triangle. The average of these two normals is then used to approximate the tangent plan at the robot's position. Here is the equation to find the vector $\vec{n}$ normal to that plane:

$$\vec{n} = \begin{pmatrix} \frac{l(-b_z + a_z + c_z - d_z)}{2} \\ \frac{l(-c_z + a_z + b_z - d_z)}{2} \\ l^2 \end{pmatrix} \tag{16}$$

where l is the width and length of one cell of the DEM, and a_z, b_z, c_z, d_z are respectively the heights of the top left, top right, bottom left and bottom right corners of the cell, considering the frame positioned as shown in Fig. 5.

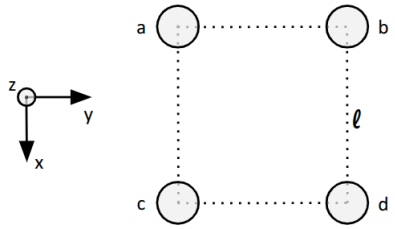

Fig. 5: DEM cell with corners labels and frame

The previous heading vector is then projected onto this new tangent plane, resulting in the projected 3D heading vector $\vec{t}_{3D,projected}$, computed using the following formula:

$$\begin{aligned} \vec{t} &= \vec{t}_{3D} - (\vec{t}_{3D} \cdot \vec{n}) \cdot \vec{n} \\ \vec{t}_{3D,projected} &= \frac{\vec{t}}{\|\vec{t}\|} \end{aligned} \tag{17}$$

The last step of the process is to apply the angular velocity. The robot needs to turn around its own z-axis, which means, in other words, around the normal vector of the surface $\vec{n}$. The Rodrigues' rotation formula is used to calculate the rotation of the robot:

$$\begin{aligned} \theta &= \omega dt \\ \vec{t}_{3D,rotated} = \vec{t}_{3D,projected} &\cos\theta + \\ (\vec{n} \times \vec{t}_{3D,projected}) &\sin\theta + \\ \vec{n}(\vec{n} \cdot \vec{t}_{3D,projected})&(1 - \cos\theta) \end{aligned} \tag{18}$$

where θ is the angle that the robot rotates by, and $\vec{t}_{3D,rotated}$ is the heading vector at the end of this iteration.

B. Cost function

The cost function ϕ mentioned at the beginning of subsection III-A is evaluated at each waypoint, characterised by its state x and the corresponding input v. The cost function is defined as a weighted sum of terms that each correspond to one objective to optimise, referred to as a *critic*. For a cost function composed of N critics, ϕ is defined as:

$$\phi(x, v) = \sum_{i=0}^{N-1} \alpha_i C_i(x, v) \tag{19}$$

where $\alpha_i \in \mathbb{R}^+$ is the relative importance given to the critic C_i. The proposed MPPI controller relies on a cost function that includes four critics: goal reaching, slope avoidance, obstacle avoidance, and traversal speed maximisation. While this selection addresses key constraints for safe and efficient navigation, it is not exhaustive for lunar application. Additional factors such as sun exposure, surface roughness, and thermal conditions shall be integrated in future extensions, but are excluded here to focus the analysis on those objectives.

Goal Critic: this critic is responsible for favouring trajectories that bring the robot closer to the goal than others. Because of the planning system explained in Subsection III-C, the algorithm does not aim to follow a predefined global path. Instead, the path planner ensures that the robot makes global progress towards the destination while balancing multiple constraints. On one hand, this critic must be flexible enough to allow the robot to deviate from the path going straight to the goal, in order to optimise the other constraints. On the other, it should become stricter as the robot nears the goal, to prevent deadlock situations if the goal point is located in a high-cost region. For instance, as illustrated in Fig. 6, the robot circles around the goal point because the cost of climbing the hill and the cost of the goal critic compensate each other.

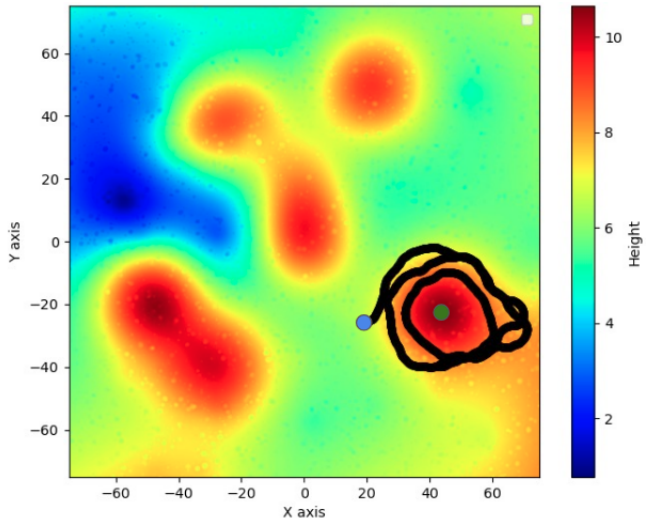

Fig. 6: Deadlock situation. The blue point is the start position, and the green point is the destination to reach.

To calculate this cost, we use the position of the last point of each trajectory to evaluate how close it gets to the goal. However, the cost cannot be based on the absolute distance to the goal. Indeed that would make this critic almost insignificant in situations where the robot is still far away from its destination. For instance, assuming it is located 100m away from the goal, one trajectory might end at 92m and another at 94m. While a 2m difference can be considered significant, the relative difference is only of around 2.15% and will not affect the trajectories weights much.

To ensure effectiveness, we propose a function that considers the closest point to the goal the robot could theoretically reach after K inputs, which will be referred to as the intermediate goal point. In other words, this involves drawing a straight line from the robot to the goal and selecting the point along that line at a distance equal to the MPPI horizon. Fig. 7 illustrates this idea, where G is the destination, and G' is the intermediate goal point:

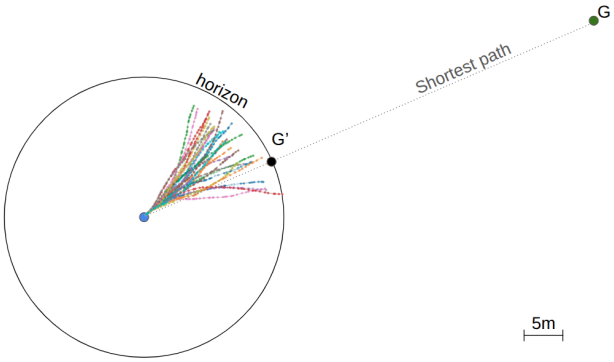

Fig. 7: Goal critic based on intermediate goal

When the goal point is not within the horizon distance yet, the cost function term C_{goal} related to the goal critic is thus computed as:

$$C_{goal} = \text{dist}(\text{Q}, G') \cdot \left(1 + \frac{2}{\text{d}}\right) \tag{20}$$

where Q is the last point of the trajectory and d is the distance between the robot and the destination. Whenever the goal enters the horizon range, the cost function changes such that all the points of the trajectory should get as close as possible to the destination:

$$C_{goal} = \sum_{t=0}^{T-1} \text{dist}(x_t, G) \tag{21}$$

Slope Avoidance Critic: Driving on sloped terrain involves several risks for the rover. Ascending a slope increases energy consumption and reduces speed, while descending requires stronger braking, which also consumes energy and may lead to situations where the rover must climb back up. More generally, navigating across inclined surfaces raises the risk of slipping, especially when executing turns.

To assess how well a trajectory remains on relatively flat terrain, a slope-related cost term is introduced. The objective

is to evaluate the slope between each pair of consecutive waypoints. Here, the use of 3D projection offers a clear advantage, as the 3D position of each waypoint has already been computed. This allows to evaluate the slope at any segment using the horizontal distance and the elevation change between two successive points.

The calculation of the exact slope angle would require the computationally expensive use of the *arctan* function. It is therefore replaced by the calculation of the ratio between the elevation change and the horizontal distance between two consecutive waypoints. This simplification holds since the *arctan* function is strictly increasing, meaning larger slopes correspond to large ratios. This approach has two advantages: it avoids the computational overhead of evaluating arctangent at every segment of every trajectory, and due to the shape of the tangent function, it naturally penalises steeper slopes more heavily than a linear cost would. Below is the equation used to compute the cost function term C_{slope} related to the steepness of a trajectory:

$$C_{slope} = \sum_{t=1}^{T} \left(1 + \left|\frac{\Delta z_t}{d_t + \epsilon}\right|\right)^2 \tag{22}$$

where Δz_t and d_t are respectively the height difference and the horizontal distance between the waypoint $t-1$ and t, and ϵ is a small term that prevents a division by 0 to occur. Note that the absolute value of the ratio is taken, since both positive and negative slopes are avoided. To further emphasise this penalisation towards slopes, each term of the sum is squared. This strongly discourages steeper slopes.

Obstacle Avoidance Critic: Avoiding collisions is a fundamental requirement in path planning. While selecting collision-free trajectories is a good start, it is even safer and more reliable to choose trajectories that stay as far away as possible from obstacles. Indeed, on challenging terrain, several factors like slippage on sandy surfaces or noise in the motors, can cause the robot to drift and/or deviate from its intended path. In cluttered areas, this increases the risk of a collision if safety measures are not taken. Additionally, by staying away from high-risk areas, the MPPI algorithm remains able to generate a larger number of valid trajectories. As explained earlier, improving the diversity of the samples leads to a better approximation of the optimal solution.

This critic relies on a costmap that quantifies collision risk. As a first step, each rock is enclosed within a sphere whose position and radius are known. The radius of each sphere is then expanded by the radius of the robot, to account for the robot's physical footprint. Any cell in the costmap that lies within one of these spheres is considered part of the collision space. In the cost function, these regions are assigned an extremely high cost, ensuring that any trajectory colliding with an obstacle is discarded. In the second step, a Gaussian-shaped cost field is added around the obstacles. It creates a gradient of increasing cost as the robot approaches an obstacle, encouraging it to keep a safe distance. The cost remains finite to allow the planner to weigh proximity against other objectives. In the example costmap shown in Fig. 8, random rocks were distributed across the area ranging from -55 to +55 in both the x and y directions.

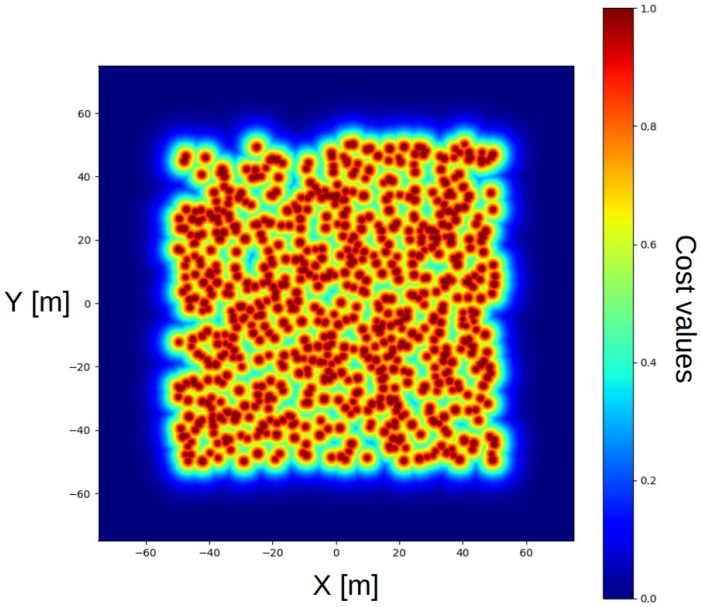

Fig. 8: Costmap quantifying collision risk

The obstacle cost C_{obs} for a trajectory of waypoints is defined as:

$$C_{obs} = \sum_{t=1}^{T} \left[I(c(x_t)) \cdot C_{hard} + c(x_t)\right] \tag{23}$$

where x_t is the t-th waypoint position of the trajectory, $c(x_t)$ is the costmap value at position x_t and C_{hard} is a very large penalty. $I(x_t)$ is a binary variable that is equal to 1 when a waypoint is inside the collision space, meaning when $c(x_t)$ is equal to 1:

$$I(c(x_t)) = \begin{cases} 1, & c(x_t) = 1 \\ 0, & c(x_t) < 1 \end{cases} \tag{24}$$

High-speed Critic: This critic is responsible for encouraging the rover to maintain a targeted high speed. At first glance, it might seem that the goal critic alone would already push the rover to move faster. Indeed, higher speeds typically bring the robot closer to the goal, resulting in lower associated costs. However, relying solely on the goal critic for speed regulation leads to a trade-off. For instance, increasing the goal critic weight to prioritise fast trajectories would also reduce the planner's flexibility to deviate from the shortest path, making it more likely to ignore slopes and to take riskier trajectories in terms of obstacle avoidance. Conversely, lowering the goal critic's weight to promote safer paths would result in slower speeds. Throughout the experiments, this imbalance even led to deadlock situations, where the MPPI chooses to reduce the velocity down to nearly zero to avoid both slopes and obstacles.

Tuning the goal critic's weight to achieve an optimal balance between safety and speed is not only difficult, but the

result is often highly dependent on the terrain topography, while not allowing to fix a certain target velocity. To address those limitations, we introduce an independent high-speed critic to the cost function. The term $C_{high-speed}$ sums the differences between the velocity at each timestep and the target velocity, giving:

$$C_{high-speed} = \sum_{t=1}^{T} |v_{target} - v_t| \tag{25}$$

where v_{target} is the targeted velocity, and v_t is the velocity of the robot at timestep t. Finally, since the robot needs to slow down when it reaches its destination, this high-speed critic term becomes equal to 0 as soon as the destination is within the horizon range of the robot.

C. Low-level controller

As will be discussed later, tests were carried out in the IsaacSim simulation environment, which offers a better degree of realism. During these tests, recurring wheel slippage was observed, causing the velocities expected by the MPPI to not always match the rover's actual motion. To mitigate this issue, a low-level PI feedback controller was introduced to reach the desired speed from MPPI. This controller compares the predicted linear and angular velocities v_p to the ones actually measured v_m, and computes a correction term u_{corr} using the following control law:

$$u_{corr} = -K_P(v_m - v_p) - K_I \int (v_m - v_p)dt \tag{26}$$

where K_P and K_I are the proportional and integral gains and dt is the timestep duration. Using an empirically identified mapping function, u_{corr} is then converted to $u_{l,corr}$ and $u_{r,corr}$ which are the input corrections for the left and right wheels. This corrective layer enables smoother and more accurate motion that aligns with MPPI predictions.

V. IMPLEMENTATION

This section details the implementation of the MPPI controller and how its parallelisation on GPU was handled. The overall workflow is summarised in the flowchart in Fig. 9, which illustrates the key steps involved in one control loop and the interactions between the CPU and GPU devices.

The MPPI controller is initialised by collecting the robot's current state, which includes its position, its heading orientation, and the wheels speed, and by loading the DEM representing its local environment. The GPU memory allocation is then performed such that all the data structures required during the control loop are available on the GPU device. Once done, the MPPI control loop will repeat as long as the task is not finished. That loop is composed of a series of GPU kernels followed by an exchange of data between the CPU and the GPU.

A. GPU Kernel Design

The control loop of the MPPI algorithm launches multiple GPU kernels in a row, each handling a distinct computational task. This modular breakdown was chosen for two reasons. First, it improves code readability and makes the implementation easier to debug, as each kernel can be tested independently from the others. Second, it allows for parallelisation across different dimensions depending on the task. Below, the role and functioning of each kernel are described, along with the corresponding pseudo-code.

`generate_inputs()`: This kernel generates sequences of inputs by sampling from the normal distribution centred on the previous optimal sequence of inputs. The sampled values are then clamped based on the inputs limitations of

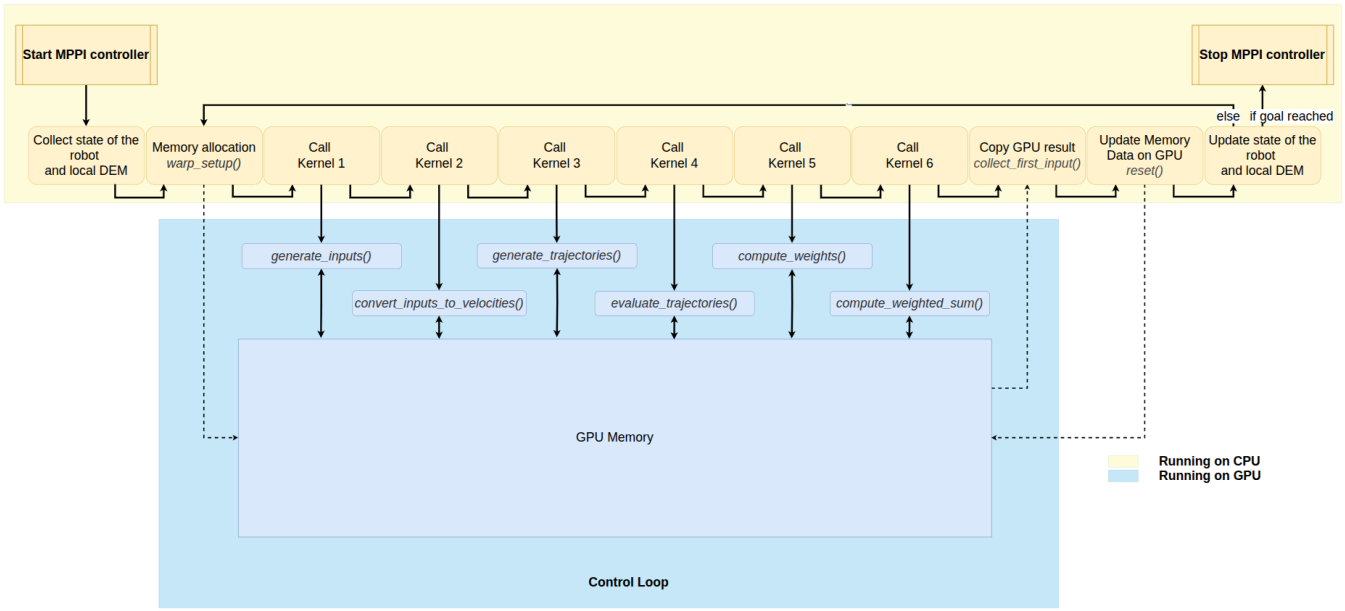

Fig. 9: Flowchart of the GPU-accelerated MPPI

the robot. As expressed in Equation 1, each input only depends on the corresponding one from the previous sequence. Consequently, every input of every trajectory can be sampled at the same time, which means the process is parallelisable over K (# trajectories) x T (# timesteps) dimensions.

Algorithm 2 generate_inputs()

Given: u_l^*, u_r^*: Previous optimal inputs sequence for the left wheel, right wheel
u_l, u_r: Generated inputs sequence for the left wheel, right wheel
ϵ_l, ϵ_r: input noise sequence for the left wheel, right wheel
$u_{\min}, u_{\max}$: Minimal, maximal input values
Σ: Covariance
$tid \leftarrow$ warp.tid()
$\epsilon_l^{tid} \leftarrow \mathcal{N}(0, \Sigma)$
$\epsilon_r^{tid} \leftarrow \mathcal{N}(0, \Sigma)$
if not_last_input **then**
  $u_l^{tid} \leftarrow \text{clamp}(u_l^{*,tid+1} + \epsilon_l^{tid}, u_{min}, u_{max})$
  $u_r^{tid} \leftarrow \text{clamp}(u_r^{*,tid+1} + \epsilon_r^{tid}, u_{min}, u_{max})$
else
  $u_l^{tid} \leftarrow \text{clamp}(u_l^{*,tid} + \epsilon_l^{tid}, u_{min}, u_{max})$
  $u_r^{tid} \leftarrow \text{clamp}(u_r^{*,tid} + \epsilon_r^{tid}, u_{min}, u_{max})$
end if

`convert_inputs_to_velocities()`: This kernel applies the forward kinematics of the system to compute the linear and angular velocities at each timestep. The wheels velocities are first determined using the update model W. These are then converted into linear and angular velocities via a differential wheel model D, which can be replaced to suit different wheeled systems. This highlights one key advantage of segmenting the control loop into kernels, and more generally of using MPPI. Since the velocity at a given timestep depends on the one from the previous timestep, parallelisation can only be performed across trajectories, i.e., over K dimensions.

Algorithm 3 convert_inputs_to_velocities()

Given: v_l, v_r: left and right wheel velocity
u_l, u_r: left and right wheel control sequences
W: update model for the wheel velocities
D: differential wheel model
v^{lin}, v^{ang}: linear, angular velocities sequences
$tid \leftarrow$ warp.tid()
$v_l, v_r \leftarrow GetVelocities()$
for $t = 0$ to $T - 1$ **do**
  $v_l \leftarrow W(v_l, u_{l,t}^{tid})$
  $v_r \leftarrow W(v_r, u_{r,t}^{tid})$
  $(v_{lin,t}^{tid}, v_{ang,t}^{tid}) \leftarrow D(v_l, v_r)$
end for

`generate_trajectories()`: This kernel is responsible for propagating each trajectory based on the sequences of linear and angular velocities previously calculated. To do this, the algorithm follows the 3D projection method described in Section IV-A. Similarly to the previous kernel, this process is iterative, meaning that a certain waypoint can only be computed once the one before is. The parallelisation is therefore in K dimensions.

Algorithm 4 generate_trajectories()

Given: q: array containing the corner heights of the robot's current DEM cell
h: height of the robot
n: normal vector of the local surface
t: heading vector of the robot
p: 2D position of the robot
X: array (KxT) of the trajectories waypoints
$tid \leftarrow$ warp.tid()
for $t = 0$ to $T - 1$ **do**
  $p^{tid} \leftarrow UpdatePositionsXY()$ ▷ Equation 15
  $q^{tid} \leftarrow GetCornersHeights()$
  $h^{tid} \leftarrow UpdatePositionZ()$
  $n^{tid} \leftarrow NormalToSurface()$ ▷ Equation 16
  $t^{tid} \leftarrow TangentToSurface()$ ▷ Equation 17
  $t^{tid} \leftarrow UpdateOrientation()$ ▷ Equation 18
  $X_t^{tid} \leftarrow array(p_0^{tid}, p_1^{tid}, h^{tid})$
end for

`evaluate_trajectories()`: Given the critics involved, this kernel assigns a cost to each trajectory. A significant portion of the tuning process focuses on this part of the code, as it determines the weight assigned to each critic. That process is parallelised over the number of trajectories.

Algorithm 5 evaluate_trajectories()

Given: S: Array containing the cost of each trajectory
ϕ: Cost function
X: array (KxT) of the trajectories waypoints
$tid \leftarrow$ warp.tid()
for $t = 0$ to $T - 1$ **do**
  $S^{tid} \leftarrow StateCost(X_t^{tid}, u_t^{tid})$ ▷ Equations 11, 19
end for

`compute_weights()`: In this kernel, the importance sampling weights are calculated based on the costs previously computed, using the aforementioned Equation 11. All the weights are computed in parallel, which means that this kernel has a dimension of K.

`compute_weighted_sum()`: Finally, the optimal control input is estimated as a weighted average of all sampled inputs. This kernel is parallelised over the number of trajectories K.

B. CPU-GPU Interaction

Memory transfers between the CPU and GPU are significantly time-consuming compared to operations performed directly on the GPU. For this reason, the control loop is

Algorithm 6 compute_weights()

Given: S: Array containing the cost of each trajectory
ω: Array of weights
ρ: Minimum cost
η: Normalisation factor
λ: Temperature
$tid \leftarrow$ warp.tid()
$\rho \leftarrow \min(S^0, S^1, \ldots, S^{K-1})$
$\eta \leftarrow \eta + \exp(-\frac{1}{\lambda}(S^{tid} - \rho))$
$\omega^{tid} \leftarrow \frac{1}{\eta}\exp(-\frac{1}{\lambda}(S^{tid} - \rho))$

Algorithm 7 compute_weighted_sum()

Given: ϵ_l, ϵ_r: input noise sequence for the left wheel, right wheel
u_l^*, u_r^*: optimal inputs sequence for the left wheel, right wheel
ω: array of weights
$tid \leftarrow$ warp.tid()
for $t = 0$ to $T - 1$ **do**
$u_l^* \leftarrow u_l^* + \omega^{tid}\epsilon_{l,t}^{tid}$
$u_r^* \leftarrow u_r^* + \omega^{tid}\epsilon_{r,t}^{tid}$
end for

implemented to minimise data exchange at each iteration. From the GPU to the CPU, only the first control input of the optimal sequence is transferred. In the other direction, the robot's position and orientation must be updated at each control loop. The local DEM also needs to be updated, but the update frequency depends on its size. Since it represents a large amount of data, updates should only occur when absolutely necessary. In fact, the DEM only needs to be refreshed when the edge of the map enters the horizon range. Otherwise, as can be seen in Fig. 10, the MPPI already has all the data it needs to operate.

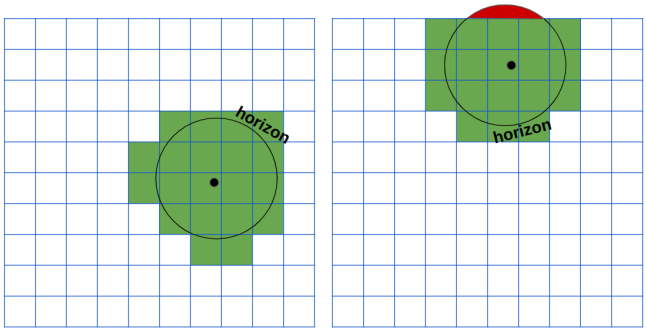

Fig. 10: Left: the height data are available for any trajectory. Right: the height data are not available for trajectories ending up in the red area.

The same goes for the costmap used for obstacle avoidance; it will be computed and updated at the same time as the DEM is.

VI. EXPERIMENTS

This section aims at evaluating the contributions presented in this work. Each subsection presents a specific experiment, including its setup, visual context, and evaluation metrics. The corresponding results are then presented in the subsequent section.

A. Validation

Before integrating the proposed model used to project trajectories in 3D into the MPPI controller, a validation step was necessary. This ensured that the mathematical formulation could accurately predict the motion of a real robot across uneven terrain.

To evaluate the reliability of the proposed projection method, a comparative experiment was conducted. The objective was to assess how closely the trajectory predicted by the algorithm matches the actual trajectory followed by a robot simulated in Gazebo, when subjected to the same sequence of control inputs and navigating the same terrain. In other words, the ground truth data was considered obtained from the Gazebo simulator [48].

The 3D projection algorithm was fully implemented in Python. As a reminder, starting from a sequence of control inputs, it calculates the corresponding linear and angular velocities using the differential wheeled robot kinematics model. These velocities are then used to simulate the robot movement over time. In this Python simulation, the robot was treated as a point. The entire process, including terrain handling, pose updates, and trajectory generation, takes place within the Python framework where the terrain was represented as a DEM. Fig. 11 shows the Python simulation with the robot represented as a red dot, and its heading angle being the red vector.

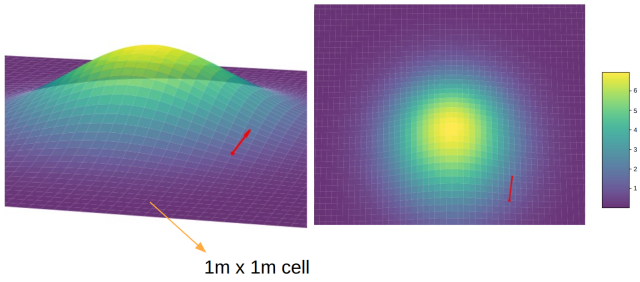

Fig. 11: Robot on uneven terrain in Python simulation

In the Gazebo simulation environment, the exact same sequence of inputs was sent to a simulated Leo Rover. This setup leveraged the Robot Operating System (ROS) to send velocity commands and record the robot's actual path as it moved across the terrain. To ensure consistency, the Python DEM was stored and then converted into a Gazebo-compatible mesh format, as shown in Fig. 12. It is important to note that, for the purpose of this comparison, a no-slip condition was enforced in the Gazebo simulation. This was done by increasing the friction coefficient of both the rover's wheels and the ground at maximum. Even though this does not reflect real-world conditions, it matched the

assumptions of the 2D and 3D projection algorithms, which do not account for wheel slippage.

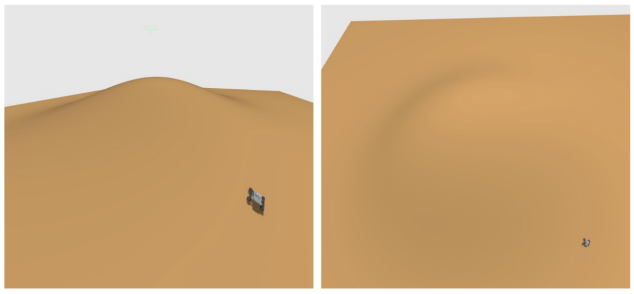

Fig. 12: Leo Rover on uneven terrain in Gazebo simulation

The test protocol was designed using two types of terrain. The first is the one shown in Fig. 11 and 12, and the second one is illustrated in Fig. 13. One features a large, prominent bump at the centre, and the other has smaller bumps randomly distributed across the surface. To ensure the robot would spend significant time on sloped terrain, it was systematically positioned either facing a slope or directly on one. This setup was essential, as the advantages of 3D over 2D trajectory prediction become apparent when navigating sloped surfaces. Indeed, on flat ground, both methods tend to produce similar results.

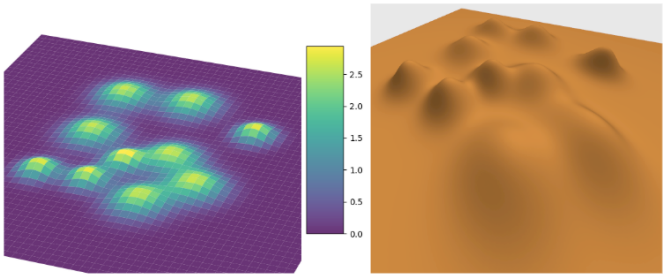

Fig. 13: Second terrain used for validation

For each terrain, 20 runs were conducted using random sequences of inputs for both the left and right wheels. Each run must be launched three times: once using the 2D projection, once using the 3D projection method proposed, and once using the simulated rover. The distance travelled was limited to $\sim$ 12m, reflecting a realistic horizon length for the MPPI algorithm.

To assess the accuracy of the 2D and 3D MPPI projections, two key metrics were considered: the orientation difference (expressed in radians), and the Euclidean distance between corresponding waypoints (expressed in meters). These metrics were calculated at each waypoint to observe how the error evolves along the trajectory. The comparison was performed separately between the simulated robot's trajectory and the one predicted by the 2D MPPI, and between the simulated trajectory and the one predicted by the 3D MPPI. To provide a clear comparison of how each approach aligns with the actual robot motion, a graph will display how the average error in both metrics grows throughout the trajectory.

B. Impact of 3d over 2d

To evaluate the impact of using 3D trajectory prediction within the MPPI control, a targeted proof-of-concept scenario was first developed. The goal is to give the reader a visualisation of how using 3D projection affects the controller behaviour. Then, a performance comparison was conducted to evaluate the benefits of the 3D approach over the traditional 2D projection when using MPPI for lunar navigation.

1) Proof of concept: In this setup, the robot started in $(0, -13)$ on flat terrain, with a prominent bump placed directly in front of it, and the destination was positioned on the opposite side of the bump in $(0, 13)$. The bump was dimensioned in such a way that if the projection is done in 2D, the shortest path directly goes through, resulting in the robot climbing over it. However when projecting the samples in 3D, then the shortest path is to go around the bump. Those two paths are illustrated in Fig. 14.

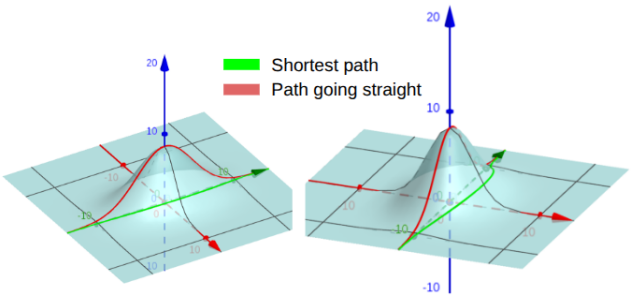

Fig. 14: Shortest path to cross a bump

This configuration allowed to assess whether incorporating 3D projections into MPPI leads to different navigation behaviours compared to the standard 2D implementation. The only critic that was used in this test was the goal critic. The result is primarily visual, as the goal was to observe whether either of the MPPI controllers decided to go around the bump to get close to the optimal path, rather than go straight and climb it up and down. To quantitatively compare both paths, the total lengths and height travelled were computed.

2) Performance comparison: The following performance comparison was done within the context of lunar navigation. As previously discussed, navigating on the Moon comes with unique challenges, such as avoiding scattered rocks and dealing with uneven and sloped terrain to maintain stability. An additional goal is to reach higher speeds, which induces more instability on sloped terrain, and a limited reaction time as well. This set of experiments aimed to measure how well each controller addresses such constraints.

The comparison was carried out entirely within a custom Python simulation. A differential wheeled robot model was used to simulate movement: control inputs were translated into linear and angular velocities, which in turn updated the position of a point representing the robot's location.

The terrain used in the simulation is based on real lunar data. The underlying topography comes from NASA's "High-Resolution LOLA Topography for Lunar South Pole Sites"

dataset, specifically the Site20v2 region [49]. This real-world terrain was then enhanced with additional features like rocks and craters. Those were randomly sampled using a Thomas point process, which allows for regions with varying densities. The realistic representation of such terrain is displayed in Fig. 15.

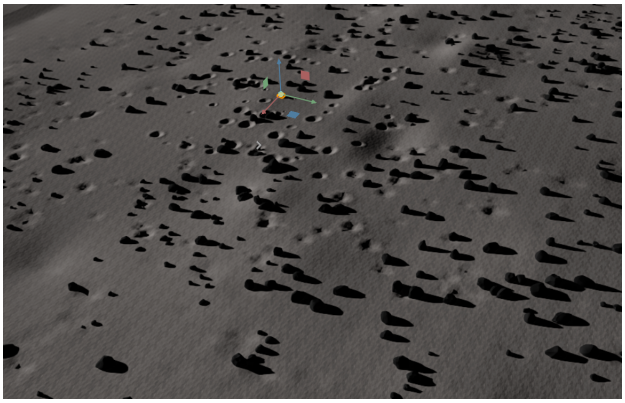

Fig. 15: Site20v2 with rocks and craters randomly sampled

The replication of this terrain in the Python simulation is represented in Fig. 16. For the sake of simplicity, the DEM that was extracted only included the elevation data corresponding to the terrain itself, meaning the sloped ground and the craters. The rocks were approximated to spheres whose position and radius are known. The slope information is visualised through a colormap. The collision space represented by the red dots is based on the rocks radius, the robot radius, and a safety margin.

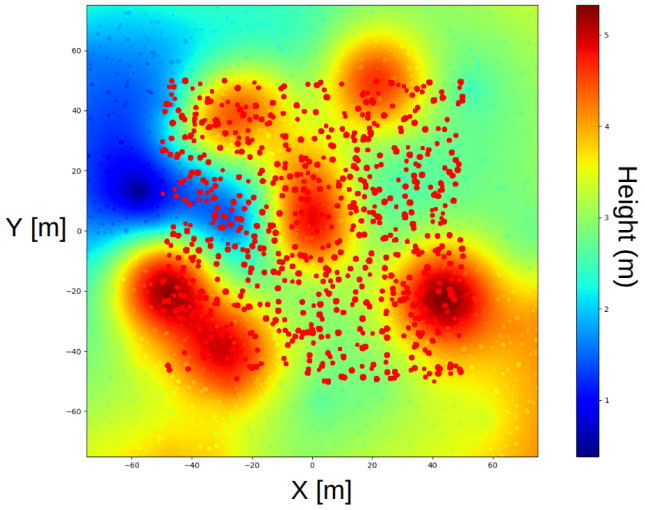

Fig. 16: Visualisation of the Python representation of Site20v2 with rocks and craters randomly sampled

To test both 2D and 3D MPPI, several configurations were used. Indeed, the goal was not only to quantify the performances, but also to determine the conditions in which they differ the most. First, the number of sampled trajectories varied between 1500, 800, and 350. Since the computational resources of a lunar rover are particularly limited, it is important to study how the MPPI reacts with fewer samples. Secondly, the runs were performed on terrains with and without additional rocks. Thirdly, the slope critic's weight was also tuned, to test both strict and soft slope avoidance strategies. For each fixed configuration, 60 navigation tasks were executed twice: one run using the 2D MPPI and one using the 3D version. In each run, start and goal positions were sampled randomly from opposite ends of the terrain, ensuring a diverse set of scenarios across the map. Such traverses have an average length of around 170m. A horizon of 100 timesteps was chosen, with a targeted speed of 2m/s, and a dt of 0.05s, resulting in a horizon length of 10m.

The performance comparison is based on four main metrics:

- the total length of the path
- the average speed
- the slope climbed up or down
- the obstacle avoidance safety

The first two criteria are quite direct to evaluate: the total length and the average speed are simply computed based on the waypoints composing the full path, and on the list of linear velocities stored from the run.

To quantify the slope avoidance performance, the slope cost term already implemented in the MPPI controller is used. Indeed, this term already estimates the slope by calculating the ratio between the height change and the horizontal distance between each waypoint. Additionally, it penalises steep slopes much more heavily than moderate ones, providing a meaningful measure of the difficulty of the terrain traversed.

The final criterion is the evaluation of obstacle avoidance safety. In high-speed navigation scenarios, since a collision can have critical or even fatal consequences, it is essential to ensure maintaining a sufficient distance from obstacles. To thoroughly evaluate the obstacle avoidance safety, two different methods are used.

The first approach relies on the cost function term used within the MPPI controller. Each waypoint is assigned a cost based on the costmap, and the total cost of a trajectory is obtained by summing the individual waypoint costs. The advantage of this evaluation method is that it is consistent with how trajectories are evaluated during the MPPI optimisation process. However, it has a notable drawback: it does not provide insight regarding the safety level at specific points along the trajectory. In other words, while it captures the overall safety, it might miss out local risky passages.

This limitation is illustrated with the example shown in Fig. 17. The trajectory obtained using 2D MPPI is green, and the one using 3D MPPI is purple.

The total costs of both trajectories are very similar. In fact, the trajectory using 2D MPPI is even rated globally slightly safer than the one using 3D MPPI. However, as highlighted in three different places, it appears that the 2D MPPI sometimes passes dangerously close to rocks because of the trajectory prediction inaccuracies. On the other hand, 3D MPPI consistently maintains a fair distance from the rocks.

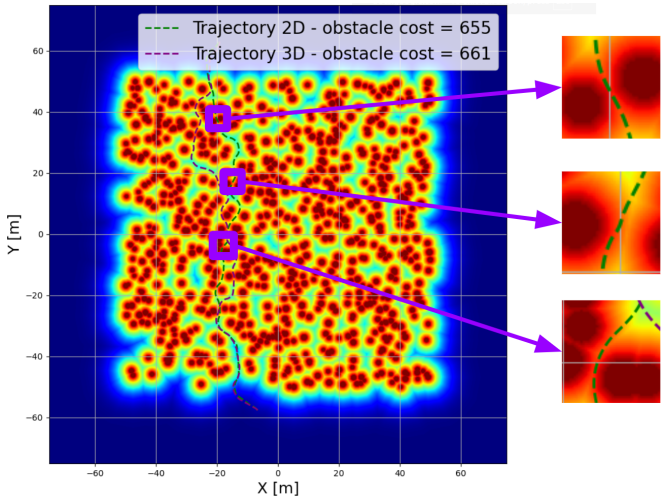

Fig. 17: Global metric missing local risk

To capture such local safety variations, a second evaluation method is introduced. Instead of relying solely on the overall trajectory cost, a table will display the number of waypoints falling within four specific distance slices from the obstacle region. The first band includes points within 10cm of the collision boundary. The second covers the next 20cm, the third the following 30cm, and the final band includes all points beyond this range. This makes it possible to clearly identify whether one controller results in trajectories that, at any point, get significantly closer to obstacles than another.

C. GPU-Implemented 3D MPPI for Lunar Navigation

This subsection presents tests related to the MPPI implementation on GPU tailored for lunar navigation. The first experiment assesses the performance improvements in control loop speed and scalability as the number of trajectories increases. Subsequently, the algorithm is validated in IsaacSim using a Husky rover simulated on real DEM data, aiming to evaluate its practical performance for high-speed navigation over challenging uneven terrain.

1) Control Loop Speed and Scalability: The key motivation behind implementing MPPI in WARP is to accelerate the control loop. All sampled trajectories are processed simultaneously across many GPU cores, allowing it to perform more efficiently than it otherwise would on regular CPU executed through languages such as C++ or Python.

Accordingly, the MPPI control loop was benchmarked across its Python, C++, and WARP-based GPU implementations. The Python implementation was coded from scratch in Python. The C++ version was the MPPI controller included in the Navigation2 stack of ROS2. That implementation has many modules running in parallel and consuming a lot of CPU resources. In order to time the MPPI control loop without being influenced by other processes running, the whole MPPI controller was ran on an isolated computer.

Each run was performed under identical conditions, using the same cost function and hyperparameters. Control loop durations are measured and compared across varying numbers of trajectories, considering both 2D and 3D projection approaches. The control loop performance of each implementation is evaluated based on execution time, measured in seconds (s).

2) Deployment in IsaacSim for Lunar Navigation: To evaluate the real-world viability of the proposed controller, experiments were carried out in the IsaacSim simulation environment using the Husky rover, a differential wheeled rover. In contrast to the Python-based simulation, where the robot's position and orientation are updated manually, the IsaacSim simulation is more physically accurate. Indeed, the MPPI directly outputs low-level wheel joint commands, which are applied to the robot's actuators. This level of realism introduces additional considerations. Since the joint commands are updated every simulation frame, the MPPI's time step (dt) had to be aligned with the loop duration of the simulator, which was empirically measured to be around 4.5 milliseconds. On top of that, an additional low-level controller, which was mentioned in subsection IV-C, was necessary to address the slippage issue. The K_P and K_I gains were tuned empirically.

Three sets of 30 runs were conducted to assess the controller performance under realistic lunar navigation conditions, each set using respectively 350, 800 and 1500 sampled trajectories. The experiments were performed on three NASA DEMs [49]: Site20v2, LM4, and LM7, illustrated in Fig. 18:

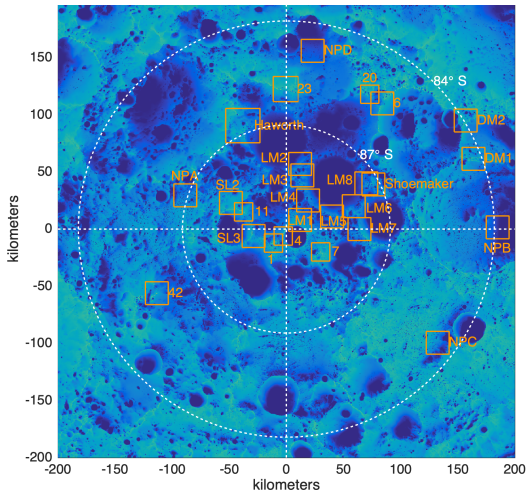

Fig. 18: Lunar South Pole Sites

To simulate the most challenging situations for the lunar rover, craters and rocks were manually added to each DEM, making it look like in Fig. 15. A random goal point was systematically set around 200m away from the start position. The target velocity was set to approximately 2m/s. For each configuration, performance was evaluated using the following metrics: goal reached (%), number of collisions, the number of times a wheel passed over a crater, and the average speed throughout one run. The results corresponding to each number of trajectories are summarised in a comparative table in the following section.

VII. RESULTS

This section presents the results obtained by conducting the experiments described in the previous section.

A. Validation

Fig. 19 displays two illustrative plots confronting the 2D and 3D trajectories with those captured in Gazebo, using the same input sequence. The terrain elevation is represented by the grey gradient circles. As one can see, the 3D and Gazebo paths deviate from the 2D one to somewhat follow the shape of the bumps.

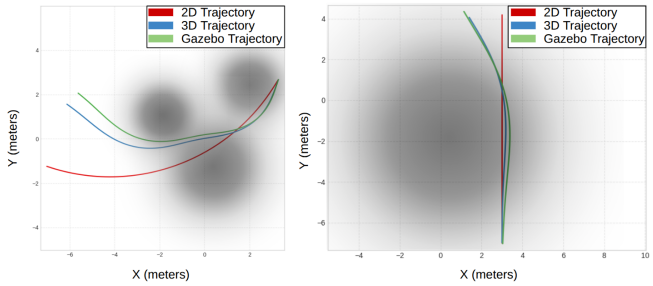

Fig. 19: 2D, 3D and Gazebo trajectories on sloped terrain

Quantitatively, the validation is based on two metrics: Euclidean distance error and orientation error. Fig. 20 and 21 summarise these errors. The horizontal axis in both graphs represents the cumulative distance travelled along the trajectory. The vertical axis in Fig. 20 shows the Euclidean distance error between a point on the 2D trajectory and the corresponding point on the Gazebo trajectory at the same travelled distance. Similarly, the vertical axis in Fig. 21 shows the orientation error between the two corresponding points.

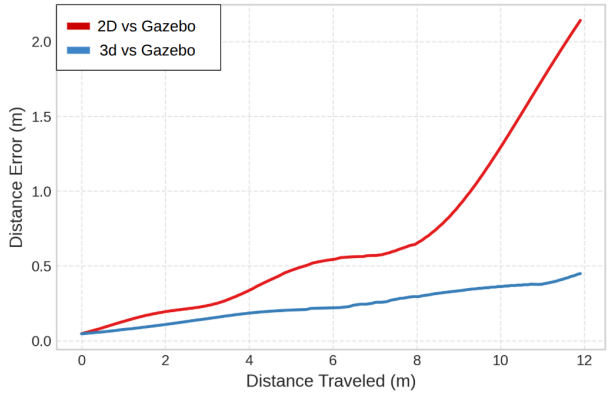

Fig. 20: Average distance error

As can be seen from the figures, both the distance and orientation errors increase more rapidly in the 2D projection compared to the 3D case. For the distance error, the 2D trajectory diverges at roughly twice the rate of the 3D trajectory over the first 6 to 8 meters. Beyond 8 meters, the error in 2D increases drastically. In terms of orientation error, the 2D projection begins to diverge noticeably after just 2 meters and shows a sharp increase after 6 meters. In contrast,

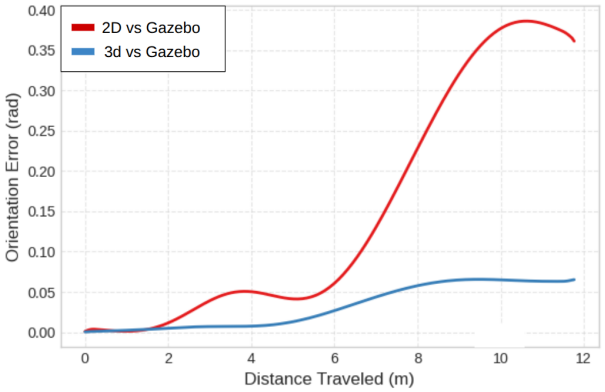

Fig. 21: Average orientation error

the orientation error for the 3D trajectory remains below 0.05 radians up to 6 meters and stays under 0.1 radians throughout the rest of the trajectory.

B. Impact of 3d over 2d

1) Proof of concept: The paths taken by both controllers are shown in Fig. 22 and 23 using a bird's-eye-view. The red trajectories represent the trajectories sampled during one loop, and the black one represents the approximated optimal trajectory during that same loop.

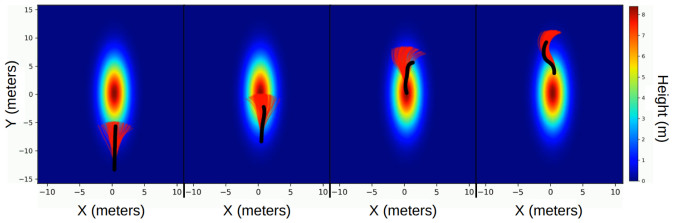

Fig. 22: 2D MPPI going straight and climbing

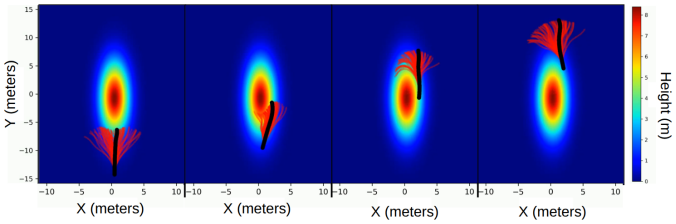

Fig. 23: 3D MPPI going around the bump

As can be seen, the MPPI controller using 2D trajectories chooses to go straight, therefore climbing the bump up and down, and taking the longer path. When using 3D trajectories, however, the controller understands that it is faster to go around the bump, as is illustrated in Fig. 14.

The second frame in Fig. 23 clearly highlights the impact of the 3D projection on the geometry. It is important to recall that those frames present a bird's-eye view, meaning the distances observed correspond only to horizontal displacement. When looking carefully one can see that the trajectories going around the bump have a longer horizontal length than

those passing through the centre. This is because the central paths involve a significant vertical ascent, which reduces the horizontal displacement viewed from above.

Below is Table I that compares the vertical distance and the total distance travelled by the 2D and 3D controllers:

Metric	MPPI 2D	MPPI 3D
Vertical distance travelled (m)	15.87	8.12
Total Distance Travelled (m)	31.32	27.98

TABLE I: Comparison of Vertical Distance and Total Distance Travelled Between MPPI 2D and MPPI 3D

The MPPI controller using 3D projection naturally chooses a path that is more optimal thanks to its accurate trajectory prediction.

2) Performance Comparison: The following tables present a performance comparison between the 2D MPPI and 3D MPPI approaches across various lunar environment configurations. Each configuration was evaluated using three different numbers of sampled trajectories, 350, 800, and 1500, to examine the impact of sampling resolution on navigation quality. The comparison is based on the following performance metrics: the total path length, the average speed, the slope avoidance, and the obstacle avoidance in terrains containing rocks.

As a point of information, the results are presented as relative performance differences rather than absolute values. Specifically, each metric will be expressed as a percentage indicating how much the 3D MPPI controller outperforms the 2D MPPI. For example, if the total path length for the 2D MPPI is 100 and for the 3D MPPI is 80, the reported result will be 20%, meaning the 3D MPPI achieved a 20% improvement over the 2D MPPI. When the 2D MPPI performs better than the 3D MPPI, the percentage will thus be negative.

Table II corresponds to the environment containing craters only. As can be seen, the percentage of improvement of 3D over 2D is around 0%, meaning that their performance was very similar. Also, the results remain constant as the number of sampled trajectories varies.

Metric	350 samples	800 samples	1500 samples
Total Path Length	-0.25%	0.01%	0.07%
Average Speed	-0.25%	0.80%	0.94%
Slope Cost	0.63%	-0.28%	1.56%

TABLE II: Performance in Crater-Only Environment

Table III reports the performance in an environment that includes both rocks and craters. In this case, the slope penalty weight in the cost function was set high, enforcing strict slope avoidance during navigation. At 350 and 1500 sampled trajectories, both 2D and 3D MPPI show similar performance. However, with 800 samples, 3D MPPI shows a slight advantage, achieving over 2.5% improvement in average speed, slope avoidance, and obstacle avoidance.

Table IV corresponds to the same environment containing both rocks and craters, but with a lower slope penalty weight,

Metric	350 samples	800 samples	1500 samples
Total Path Length	-0.36%	0.40%	0.14%
Average Speed	1.33%	**3.89%**	0.75%
Slope Cost	1.22%	**2.50%**	0.67%
Obstacle Cost	3.00%	**2.89%**	1.49%

TABLE III: Performance in Rock and Crater Environment with Strict Slope Avoidance

allowing softer slope avoidance behaviour. This setting enables the rover to occasionally take small slopes to optimise for other navigation factors. In this setup, the 3D MPPI outperforms the traditional one independently of the number of samples. Notably, it achieves over 3.8% higher average speed and more than 3.5% better obstacle avoidance.

Metric	350 samples	800 samples	1500 samples
Total Path Length	0.53%	-0.18%	0.71%
Average Speed	**3.80%**	**4.92%**	**4.02%**
Slope Cost	2.18%	0.4%	3.19%
Obstacle Cost	**4.59%**	**3.57%**	**3.74%**

TABLE IV: Performance in Rock and Crater Environment with Soft Slope Avoidance

To evaluate obstacle avoidance safety more precisely, the second metric that was presented is used. This method involves quantifying how many waypoints, across a total of 60 trajectories, fall within distance slices surrounding obstacles. As a reminder, the first slice corresponds to the point being withing 10cm from the collision boundary, the second covers the next 20cm, the third the following 30cm, and the fourth includes the remaining points. In order to provide the reader with more interpretability, note that 100 waypoints represent a duration of $100dt = 5s$. In other words, 100 waypoints located in the first slice means that the rover spent 5s in the closest area to the collision space.

Table V shows the distribution of waypoints across the four defined slices for both 2D and 3D trajectory projections, at three different sampling levels (350, 800, and 1500 trajectories). Those results correspond to the case where strict slope avoidance is selected. The number of waypoints in the closest region using 2D MPPI consistently exceeds that of the 3D MPPI, with a difference of more than 15% for any number of sampled trajectories.

Slice Region	350 trajectories	800 trajectories	1500 trajectories
Slice 1 (2D)	**365**	**190**	**246**
Slice 1 (3D)	**283**	**164**	**210**
Slice 2 (2D)	4136	3524	4045
Slice 2 (3D)	4275	2775	3905
Slice 3 (2D)	16206	14348	14838
Slice 3 (3D)	15617	13001	15317
Slice 4 (2D)	84769	80332	81633
Slice 4 (3D)	85304	77176	81234

TABLE V: Obstacle Avoidance Safety (with strict slope avoidance)

When softening the slope avoidance, the following results observed are shown in Table VI. In this case, the previously observed trend is even more pronounced, with the 2D MPPI

generating over 25% more waypoints in the closest region to obstacles compared to the 3D MPPI. In addition to that, the 2D MPPI also has significantly more waypoints than the 3D MPPI in the second slice.

Slice Region	350 trajectories	800 trajectories	1500 trajectories
Slice 1 (2D)	**202**	**182**	**191**
Slice 1 (3D)	**163**	**89**	**112**
Slice 2 (2D)	**3114**	**2623**	**2655**
Slice 2 (3D)	**2887**	**2171**	**2299**
Slice 3 (2D)	12902	13501	13178
Slice 3 (3D)	12542	12939	12905
Slice 4 (2D)	85342	82741	83124
Slice 4 (3D)	85169	83614	83076

TABLE VI: Obstacle Avoidance Safety (with soft slope avoidance)

C. GPU-Implemented 3D MPPI Performance for Lunar Navigation

1) Control Loop Speed and Scalability: To evaluate and compare the computational efficiency of the different MPPI implementations, the average execution time per control loop was measured for each of them. Table VII presents those durations in seconds:

Implementation	350 samples	800 samples	1250 samples	1500 samples
CPU Python (2D)	0.25	0.53	0.86	1.45
CPU C++ (2D)	0.021	0.031	0.049	0.091
GPU WARP (2D)	0.0021	0.0024	0.0027	0.0034
GPU WARP (3D)	0.0022	0.0025	0.0029	0.0037

TABLE VII: Average Control Loop Execution Time (s)

Looking at those results, one order of magnitude separates each programming language. This demonstrates that the GPU-based implementation significantly outperforms the CPU-based MPPI in terms of computational efficiency. The difference between the MPPI controller using 2D and 3D projections is less than 0.3ms.

In addition to that, the control loop duration increases significantly faster with the number of trajectories in the CPU implementations. Specifically, it becomes more than 4 times greater in Python and C++ when going from 350 to 1500 samples. In GPU, however, it scales much more efficiently, increasing of a bit more than 1.5.

2) Deployment in IsaacSim for Lunar Navigation: The navigation performance of the Husky rover over 30 runs in various lunar environments, using 350, 800 and 1500 sampled trajectories, is summarised in Table VIII.

Metric	350 trajectories	800 trajectories	1500 trajectories
Goal Reached (%)	89.12	94.57	97.21
Collisions	2	0	0
Crater Traversals	14	9	8
Avg Speed (m/s)	1.74	1.85	1.83

TABLE VIII: Lunar Navigation Performance in IsaacSim

In terms of computational efficiency, no lag in IsaacSim was observed, indicating a high-control loop rate. Given that the controller operates synchronously with the simulator, any computation delays would have resulted in a frozen or stuttering simulation. The absence of such issues demonstrates the effectiveness of the proposed implementation for high-frequency real-time deployment.

VIII. Discussions

This section discusses the experimental findings and reflects on their implications. The first subsection focuses on the impact of the 3D trajectory projection on the MPPI performance. The second one addresses the considerations regarding the GPU-based MPPI controller for lunar navigation application. The third one discusses the limitations of the contributions and the ways of improvements.

A. MPPI with 3D Trajectories Prediction

The 3D trajectory projection method was validated against simulated rover behaviour, demonstrating its ability to approximate the robot's motion in Gazebo with greater fidelity than 2D methods. Significant errors in Euclidean distance and orientation were observed when using 2D trajectories. The cumulative yaw deviation led to a growing offset, reaching in average 50 cm at a 5-meter horizon. Beyond that distance, the difference keeps increasing even faster. Since 50cm is an average, it is important to mention that this error exceeds this value in many cases. Although one might argue that the controller is able to compensate for trajectory prediction errors over time, high-speed motion limits the vehicle's manoeuvrability. This can reduce the freedom to respond in the event that an unexpected obstacle enters in the sampling space because of some trajectory prediction error.

The performance benefits of using 3D projections were found to be dependent on terrain characteristics. In crater-only environments, 3D and 2D MPPI showed similar performance regardless of the number of trajectories. This can be explained by the fact that the absence of rocks removes the trade-off between avoiding obstacles and optimising for slope. The controller can simply choose the flattest terrain, which results in similar predictions for both 2D and 3D MPPI.

In terrains containing both rocks and craters, with strict slope avoidance, performances remained relatively similar. However, an improvement was observed with the 3D MPPI in terms of average speed and obstacle avoidance when using 800 samples. This behaviour can be interpreted as follows: with 350 samples, both controllers lacked sufficient data to identify optimal trajectories, resulting in similar suboptimal trajectories for both. With 1500 samples, the convergence toward optimal solutions allowed the 2D controller to balance the predictions inaccuracies, and perform similarly to the 3D MPPI. With 800 samples, however, a tipping point was found. The inaccuracies of 2D projections, combined with the lower sample count, prevented it from following trajectories as good as the 3D MPPI, leading to poorer performances in average speed, slope management and obstacle avoidance safety.

When slope avoidance was softened, the benefits of 3D MPPI became more pronounced. As the rover drove more on sloped terrain, the trajectory prediction error of 2D MPPI

increased, leading to less safe trajectories, as evidenced by reduced obstacle avoidance performance. To compensate for this, the 2D controller reduced speed to return to safer paths. In contrast, the 3D planner was able to follow safer trajectories, allowing it to maintain a higher average speed.

On top of that, the second approach to compare obstacle avoidance safety revealed a critical performance gap between 2D and 3D MPPI. The 2D MPPI significantly spent more time in the high-risk zone near obstacles. This phenomenon became even more pronounced when the slope avoidance was softened. In such cases, the 2D MPPI chose paths with a largely higher number of waypoints in the two closest regions to obstacles, regardless of the number of sampled trajectories.

The overall analysis shows that traditional 2D MPPI is inherently robust to uneven ground and to the resulting trajectory prediction inaccuracies. However, some scenario were highlighted where the 3D MPPI outperforms it, in particular when the robot must navigate sloped terrain filled with rocks. The 3D MPPI indeed manages to keep a safer distance from the collision space than the 2D MPPI whose inaccuracies lead to dangerous passages. An interesting idea for improvement would be to dynamically switch from 2D to 3D projections, for instance based on IMU data, when the robot detects that it is being forced to drive on significantly uneven terrain.

B. GPU-Based MPPI for Lunar Navigation

The GPU-accelerated MPPI controller developed using the WARP framework was benchmarked against Python and C++ implementations. The control loop performance clearly highlighted the acceleration provided by the GPU version. In particular, increasing the number of sampled trajectories significantly impacted the control loop durations of the CPU-based versions, whereas the GPU-based MPPI scaled much more efficiently. This result demonstrates the potential of parallelisation for real-time trajectory optimisation in sampled-based MPPI, particularly in high-speed applications that demand high-frequency control updates.

The GPU-based MPPI controller, combined with the designed cost function, consistently achieved high success rates in reaching the navigation goal. Two collisions were recorded when using 350 sampled trajectories, caused by unanticipated last-minute deviations due to terrain slippage. However, no collisions occurred with higher sample counts, showing that richer sampling indeed allows for more optimal, and therefore safer, trajectories.

Failures to reach the goal generally stemmed from deadlock situations, where the placement of rocks prevented the MPPI controller from sampling any safe trajectories, forcing the robot to stop. Aside from these edge cases, the controller maintained safe navigation across all tested terrain configurations. It avoided most craters, travelled at high speeds, and rarely crossed unsafe regions.

C. Limitations & Future work

Despite the promising results, several limitations must be acknowledged within the framework of this research. Firstly, the 3D projection method, though effective, relies on a simple model to predict linear and angular velocities. That model ignores the friction factor, which can lead, for instance, to unexpected slippage. The current model also does not account for factors that could reduce velocity under the same control inputs, such as ascending slopes or varying surface roughness. A promising mitigation strategy is the use of a low-level PI feedback controller, as presented in this work to correct speed mismatches.

Throughout the research, the assumptions of perfect perception and perfect localisation were maintained. In practice, however, those processes are real challenges, especially in the context of lunar navigation. In the scope of this thesis, they were deliberately excluded to focus on the motion planning area.

Lastly, while the GPU-based MPPI demonstrates strong potential for onboard, high-frequency motion planning, its real-world deployment in space remains constrained by current hardware limitations. GPUs are not yet radiation-hardened, although GPU-like architectures are being developed for such applications [50, 51]. Nevertheless, although the primary intended application of this research is lunar navigation, the insights gained regarding the parallelisation potential of MPPI remain broadly applicable, among others to Earth-based navigation tasks.

A natural continuation of this work involves validating the MPPI motion planner in a real-world setting. For instance, the SpaceR research group in Luxembourg, where this research was conducted, operates a dedicated lunar analogue environment along with planetary rovers. Testing MPPI in this environment would provide valuable insights into the planner's robustness under realistic conditions, in order to fill the sim-to-real gap. Furthermore, real-world testing will likely require further tuning, particularly of the cost function weights. This reflects a broader challenge in MPPI, which is that performance is closely tied to the parameters tuning. In this context, learning-based methods could enhance the controller's adaptability across varying terrain types.

Another important direction is the integration of the motion planner into a complete robotic autonomy pipeline. In future work, proper perception and localisation systems should be incorporated to evaluate MPPI's performance within the constraints of a real robotic stack. Additionally, simulating degraded sensor performance to approximate lunar conditions would provide further insights of the MPPI planner suitability for planetary navigation missions.

IX. CONCLUSION

This work explores the use of MPPI for fast and autonomous navigation on the Moon, focusing on two main aspects: a GPU-based implementation using the WARP framework to enable large-scale parallel sampling, and a 3D trajectory projection model to better account for lunar topography in motion planning. The results indicate that GPU-accelerated MPPI is a promising solution for such application. While further testing in real-world conditions is required, its ability to handle multiple constraints efficiently,

combined with the scalability of GPU parallelism, makes it well-suited for future lunar missions. Finally, although 3D trajectory prediction improves performance in specific cases, traditional 2D MPPI remains highly robust. The added computational cost of 3D planning may only be justified in scenarios involving significant terrain variations, suggesting a hybrid strategy that switches between 2D and 3D projections could be optimal.

REFERENCES

[1] Andrew J. Butrica. "To See the Unseen: A History of Planetary Radar Astronomy". In: (1996). Archived from the original on August 23, 2007. URL: https://ntrs.nasa.gov/.

[2] Matteo De Benedetti et al. "RAPID FASTNAV Projects: High-Speed Semi-Autonomous Rovers Enabling High Return Planetary Missions". In: (2024), pp. 260–265. DOI: 10.1109/iSpaRo60631.2024.10688088.

[3] A. Colaprete et al. "An Overview of the Volatiles Investigating Polar Exploration Rover (VIPER) Mission". In: 2019 (2019), P34B–03.

[4] J. Ricardo Sánchez, Martin Azkarate, and Carlos J. Pérez-del-Pulgar. "Multi-scale Path Planning for a Planetary Exploration Vehicle with Multiple Locomotion Modes". In: (2018). Oral presentation. URL: mailto:ricardosan@uma.es.

[5] Matteo De Benedetti et al. "RAPID FASTNAV Projects: High-Speed Semi-Autonomous Rovers Enabling High Return Planetary Missions". In: (2024), pp. 260–265. DOI: 10.1109/iSpaRo60631.2024.10688088.

[6] Toshiki Tanaka and Heidar Malki. "A Deep Learning Approach to Lunar Rover Global Path Planning Using Environmental Constraints and the Rover Internal Resource Status". In: *Sensors* 24.3 (2024). ISSN: 1424-8220. DOI: 10.3390/s24030844. URL: https://www.mdpi.com/1424-8220/24/3/844.

[7] Tyler Han et al. "Model Predictive Control for Aggressive Driving Over Uneven Terrain". In: *arXiv preprint arXiv:2311.12284* (2024). Accepted to R:SS 2024. URL: https://doi.org/10.48550/arXiv.2311.12284.

[8] Grady Williams et al. "Aggressive driving with model predictive path integral control". In: (2016), pp. 1433–1440. DOI: 10.1109/ICRA.2016.7487277.

[9] Grady Williams et al. "Information Theoretic Model Predictive Control: Theory and Applications to Autonomous Driving". In: *arXiv preprint arXiv:1707.02342* (2017). URL: https://arxiv.org/abs/1707.02342.

[10] Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. "Model Predictive Path Integral Control: From Theory to Parallel Computation". In: *Journal of Guidance, Control, and Dynamics* 40.2 (2017), pp. 344–357. DOI: 10.2514/1.G001921.

[11] J. Betz et al. "Autonomous Vehicles on the Edge: A Survey on Autonomous Vehicle Racing". In: *IEEE Open Journal of Intelligent Transportation Systems* 3 (2022), pp. 458–488.

[12] Miles Macklin. "Warp: Differentiable Spatial Computing for Python". In: SIGGRAPH Courses '24 (2024). DOI: 10.1145/3664475.3664543. URL: https://doi.org/10.1145/3664475.3664543.

[13] Antoine Richard et al. "OmniLRS: A Photorealistic Simulator for Lunar Robotics". In: *arXiv preprint arXiv:2309.08997* (2023). URL: https://doi.org/10.48550/arXiv.2309.08997.

[14] I.E. Gargano, K.D. von Ellenrieder, and M. Vivolo. "A Survey of Trajectory Planning Algorithms for Off-Road Uncrewed Ground Vehicles". In: Lecture Notes in Computer Science 14615 (2024). Ed. by J. Mazal et al., pp. 112–126. DOI: 10.1007/978-3-031-71397-2_8. URL: https://doi.org/10.1007/978-3-031-71397-2_8.

[15] D. Fox, W. Burgard, and S. Thrun. "The dynamic window approach to collision avoidance". In: *IEEE Robotics Automation Magazine* 4.1 (1997), pp. 23–33. DOI: 10.1109/100.580977.

[16] O. Khatib. "Real-time obstacle avoidance for manipulators and mobile robots". In: 2 (1985), pp. 500–505. DOI: 10.1109/ROBOT.1985.1087247.

[17] Paolo Fiorini and Zvi Shiller. "Motion Planning in Dynamic Environments Using Velocity Obstacles". In: *The International Journal of Robotics Research* 17.7 (1998), pp. 760–772. DOI: 10.1177/027836499801700706. eprint: https://doi.org/10.1177/027836499801700706. URL: https://doi.org/10.1177/027836499801700706.

[18] José Ricardo Sánchez-Ibáñez, Carlos J. Pérez-del-Pulgar, and Alfonso García-Cerezo. "Path Planning for Autonomous Mobile Robots: A Review". In: *Sensors* 21.23 (2021). Submission received: 16 October 2021 / Revised: 22 November 2021 / Accepted: 23 November 2021 / Published: 26 November 2021, p. 7898. DOI: 10.3390/s21237898. URL: https://doi.org/10.3390/s21237898.

[19] Christian Henkel, Alexander Bubeck, and Weiliang Xu. "Energy Efficient Dynamic Window Approach for Local Path Planning in Mobile Service Robotics**This work was conducted at the University of Auckland, Auckland, New Zealand". In: *IFAC-PapersOnLine* 49.15 (2016). 9th IFAC Symposium on Intelligent Autonomous Vehicles IAV 2016, pp. 32–37. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2016.07.610. URL: https://www.sciencedirect.com/science/article/pii/S2405896316308813.

[20] Li Xie et al. "Power-minimization and energy-reduction autonomous navigation of an omnidirectional Mecanum robot via the dynamic window

approach local trajectory planning". In: *International Journal of Advanced Robotic Systems* 15.1 (2018), p. 1729881418754563. DOI: 10.1177/1729881418754563. URL: https://doi.org/10.1177/1729881418754563.

[21] B. Tang et al. "Path planning based on improved hybrid A* algorithm". In: *Journal of Advanced Computational Intelligence and Intelligent Informatics* 25.1 (2021), pp. 64–72. DOI: 10.20965/jaciii.2021.p0064.

[22] S. Karaman and E. Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *International Journal of Robotics Research* 30.7 (2011), pp. 846–894. DOI: 10.1177/0278364911418381.

[23] Christos Katrakazas et al. "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions". In: *Transportation Research Part C: Emerging Technologies* 60 (2015), pp. 416–442.

[24] Anis Said et al. "Local trajectory planning for autonomous vehicle with static and dynamic obstacles avoidance". In: (2021), pp. 410–416.

[25] Kai Hu and Kun Cheng. "Trajectory planning for an articulated tracked vehicle and tracking the trajectory via an adaptive model predictive control". In: *Electronics* 12.9 (2023), p. 1988.

[26] Jiachen Liu et al. "A nonlinear model predictive control formulation for obstacle avoidance in high-speed autonomous ground vehicles in unstructured environments". In: *Vehicle System Dynamics* 56.6 (2018), pp. 853–882.

[27] Nicolas Goulet and Biruk Ayalew. "Energy-optimal ground vehicle trajectory planning on deformable terrains". In: *IFAC-PapersOnLine* 55.27 (2022), pp. 196–201.

[28] Y. Kuwata. "Trajectory Planning for Unmanned Vehicles Using Robust Receding Horizon Control". In: (2007). URL: https://dspace.mit.edu/handle/1721.1/38643.

[29] Taekyung Kim et al. "Smooth Model Predictive Path Integral Control Without Smoothing". In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 10406–10413. DOI: 10.1109/LRA.2022.3192800.

[30] Stepan Dergachev, Kirill Muravyev, and Konstantin Yakovlev. "2.5D Mapping, Pathfinding and Path Following For Navigation Of A Differential Drive Robot In Uneven Terrain". In: *IFAC-PapersOnLine* 55.38 (2022). 13th IFAC Symposium on Robot Control SYROCO 2022, pp. 80–85. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2023.01.137. URL: https://www.sciencedirect.com/science/article/pii/S2405896323001441.

[31] Samuel Triest et al. "Learning Risk-Aware Costmaps via Inverse Reinforcement Learning for Off-Road Navigation". In: *arXiv preprint arXiv:2302.00134* (2023). URL: https://doi.org/10.48550/arXiv.2302.00134.

[32] Masafumi Endo, Kohei Honda, and Genya Ishigami. "BenchNav: Simulation Platform for Benchmarking Off-road Navigation Algorithms with Probabilistic Traversability". In: (2024). URL: https://doi.org/10.48550/arXiv.2405.13318.

[33] Hojin Lee, Junsung Kwon, and Cheolhyeon Kwon. "Learning-based Uncertainty-aware Navigation in 3D Off-Road Terrains". In: *arXiv preprint arXiv:2209.09177* (2022). URL: https://doi.org/10.48550/arXiv.2209.09177.

[34] Bogdan Vlahov et al. "MPPI-Generic: A CUDA Library for Stochastic Trajectory Optimization". In: *arXiv preprint arXiv:2409.07563* (2024). Submitted on 11 Sep 2024, last revised 10 Mar 2025.

[35] Aniket Datar, Chenhui Pan, and Xuesu Xiao. "Learning to Model and Plan for Wheeled Mobility on Vertically Challenging Terrain". In: *arXiv preprint arXiv:2306.11611* (2023). arXiv: 2306.11611 [cs.RO]. URL: https://arxiv.org/abs/2306.11611.

[36] David D. Fan et al. "STEP: Stochastic Traversability Evaluation and Planning for Risk-Aware Off-Road Navigation". In: *arXiv preprint arXiv:2103.02828* (2021). arXiv: 2103.02828 [cs.RO]. URL: https://arxiv.org/abs/2103.02828.

[37] Philipp Krüsi et al. "Driving on Point Clouds: Motion Planning, Trajectory Optimization, and Terrain Assessment in Generic Nonplanar Environments". In: *Journal of Field Robotics* 34.5 (2017), pp. 940–984. DOI: 10.1002/rob.21694.

[38] Siyuan Yu, Congkai Shen, and Tulga Ersal. "Nonlinear Model Predictive Planning and Control for High-Speed Autonomous Vehicles on 3D Terrains". In: *IFAC-PapersOnLine* 54.20 (2021). Modeling, Estimation and Control Conference MECC 2021, pp. 412–417. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2021.11.208. URL: https://www.sciencedirect.com/science/article/pii/S2405896321022527.

[39] Brian César-Tondreau et al. "Towards Fully Autonomous Negative Obstacle Traversal via Imitation Learning Based Control". In: *Robotics* 11.4 (2022), p. 67. DOI: 10.3390/robotics11040067.

[40] Jason Gibson et al. "A Multi-step Dynamics Modeling Framework for Autonomous Driving in Multiple Environments". In: *arXiv preprint arXiv:2305.02241* (2023). arXiv: 2305.02241 [cs.RO]. URL: http://arxiv.org/abs/2305.02241.

[41] Xuesu Xiao, Joydeep Biswas, and Peter Stone. "Learning Inverse Kinodynamics for Accurate High-Speed Off-Road Navigation on Unstructured Terrain". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 6054–6060. DOI: 10.1109/LRA.2021.3073808.

[42] Hojin Lee et al. “Learning Terrain-Aware Kinodynamic Model for Autonomous Off-Road Rally Driving With Model Predictive Path Integral Control”. In: *IEEE Robotics and Automation Letters* (2023). Accepted to IEEE Robotics and Automation Letters (and ICRA 2024). DOI: 10.1109/LRA.2023.3318190. arXiv: 2305.00676 [cs.RO]. URL: https://doi.org/10.48550/arXiv.2305.00676.

[43] Corrado Pezzato et al. “Sampling-based Model Predictive Control Leveraging Parallelizable Physics Simulations”. In: *arXiv preprint arXiv:2307.09105* (2023). Submitted to RA-L. Code and videos available at https://doi.org/10.48550/arXiv.2307.09105.

[44] Evangelos A. Theodorou and Emanuel Todorov. “Relative Entropy and Free Energy Dualities: Connections to Path Integral and KL Control”. In: (2012), pp. 1466–1473.

[45] S. Dergachev and K. Yakovlev. “Model predictive path integral for decentralized multi-agent collision avoidance”. In: *PeerJ Computer Science* 10 (2024), e2220. DOI: 10.7717/peerj-cs.2220. URL: https://doi.org/10.7717/peerj-cs.2220.

[46] Gwendolyn D. Bart and H.J. Melosh. “Distributions of boulders ejected from lunar craters”. In: *Icarus* 209.2 (2010), pp. 337–357. ISSN: 0019-1035. DOI: https://doi.org/10.1016/j.icarus.2010.05.023. URL: https://www.sciencedirect.com/science/article/pii/S0019103510002150.

[47] Xiaoqiang Yu, Ping Wang, and Zexu Zhang. “Learning-Based End-to-End Path Planning for Lunar Rovers with Safety Constraints”. In: *Sensors* 21.3 (2021). ISSN: 1424-8220. URL: https://www.mdpi.com/1424-8220/21/3/796.

[48] N. Koenig and A. Howard. “Design and use paradigms for Gazebo, an open-source multi-robot simulator”. In: 3 (2004), 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[49] M.K. Barker et al. “Improved LOLA Elevation Maps for South Pole Landing Sites: Error Estimates and Their Impact on Illumination Conditions”. In: *Planetary and Space Science* 203 (Sept. 2021), p. 105119. DOI: 10.1016/j.pss.2020.105119.

[50] Wesley Powell et al. “Commercial Off-The-Shelf GPU Qualification for Space Applications”. In: (Sept. 2018). GSFC-E-DAA-TN61440, Public Distribution. URL: https://ntrs.nasa.gov/citations/20180006906.

[51] Leonidas Kosmidis et al. “GPU4S: Embedded GPUs in Space – Latest Project Updates”. In: *Microprocessors and Microsystems* 80 (2020). Also available as arXiv:2109.11074, p. 103143. DOI: 10.1016/j.micpro.2020.103143. URL: https://doi.org/10.48550/arXiv.2109.11074.