Model Predictive Path Integral Control Framework for Partially Observable Navigation: A Quadrotor Case Study

Ihab S. Mohamed, Guillaume Allibert, Philippe Martinet

Model Predictive Path Integral Control Framework for Partially Observable Navigation: A Quadrotor Case Study

Ihab S. Mohamed[1] and Guillaume Allibert[2] and Philippe Martinet[1]

***Abstract*— Recently, Model Predictive Path Integral (MPPI) control algorithm has been extensively applied to autonomous navigation tasks, where the cost map is mostly assumed to be known and the 2D navigation tasks are only performed. In this paper, we propose a generic MPPI control framework that can be used for 2D or 3D autonomous navigation tasks in either fully or partially observable environments, which are the most prevalent in robotics applications. This framework exploits directly the 3D-voxel grid acquired from an on-board sensing system for performing collision-free navigation. We test the framework, in realistic RotorS-based simulation, on goal-oriented quadrotor navigation tasks in a cluttered environment, for both fully and partially observable scenarios. Preliminary results demonstrate that the proposed framework works perfectly, under partial observability, in 2D and 3D cluttered environments.**

MULTIMEDIA MATERIAL

The supplementary video attached to this work is available at: https://bit.ly/2PAbESO

I. INTRODUCTION

Having a safe and reliable system for autonomous navigation of robotic systems such as Unmanned Aerial Vehicles (UAVs) is a highly challenging and partially-solved problem for robotics communities, especially for cluttered and GPS-denied environments such as dense forests, crowded offices, corridors, and warehouses. Such a problem is very important for solving many complex applications, such as surveillance, search-and-rescue, and environmental mapping. To do so, UAVs should be able to navigate with complete autonomy while avoiding all kinds of obstacles in real-time. To this end, they must be able to (i) perceive their environment, (ii) understand the situation they are in, and (iii) react appropriately.

Obviously enough, this problem has been already addressed in the literature, particularly those works related to dynamics and control, motion planning, and trajectory generation in unstructured environments with obstacles [1], [2], [3], [4], [5]. Moreover, the applications of the path-integral control theory have recently become more prevalent. One of the most noteworthy works is Williams's iterative path integral method, namely MPPI control framework [6]. In this method, the control sequence is iteratively updated to obtain the optimal solution on the basis of importance sampling of trajectories. In [7], authors derived a different iterative method in which the control- and noise-affine dynamics constraints, on the original MPPI framework, are eliminated. This framework is mainly based on the information-theoretic interpretation of optimal control using KL-divergence and free energy, while it was previously based on the linearization of Hamilton-Jacob Bellman (HJB) equation and application of Feynman-Kac lemma. Although different methods are adopted to derive the MPPI framework, they are practically equivalent and theoretically related[1]. An extension to Williams's information-theoretic-based approach is presented in [8], where a learned model is used to generate informed sampling distributions.

The attractive features of MPPI controller, over alternative methods, can be summarized as: (i) a derivative-free optimization method, i.e. no need for derivative information to find the optimal solution; (ii) no need for approximating the system dynamics and cost functions with linear and quadratic forms, i.e., non-linear and non-convex functions can be naturally employed, even that dynamics and cost models can be easily represented using neural networks; (iii) planning and execution steps are combined into a single step, providing an elegant control framework for autonomous vehicles. However, one drawback of MPPI is that its convergence rate is empirically slow, which has been addressed in [9].

In the context of autonomous navigation, it is observed that the MPPI controller has been mainly applied to the tasks of aggressive driving and UAVs navigation in 2D cluttered environments. To do so, MPPI requires a cost map, as an environment representation, to drive the autonomous vehicle. Concerning autonomous driving, the cost map is obtained either *off-line* [7] or from an on-board monocular camera using deep learning approaches [10], [11]. Regarding UAV navigation in cluttered environments, the obstacle (i.e., cost) map is assumed to be available, and only static 2D floor-maps are used. Conversely, in practice, the real environments are often partially observable, with dynamic obstacles. Moreover, it is noteworthy that only 2D navigation tasks are performed so far, which limits the applicability of the control framework. For this reason, this paper focuses on MPPI for 2D and 3D navigation tasks in a previously unseen and dynamic environment. In particular, the main contributions of our work can be summarized as follows:

This research was supported by the ANR CLARA project (ANR-18-CE33-0004).

[1]Ihab S. Mohamed and Philippe Martinet are with the Université Côte d'Azur, Inria, France, {ihab.mohamed, philippe.martinet}@inria.fr

[2]Guillaume Allibert is with the Université Côte d'Azur, CNRS, I3S, France. allibert@unice.fr

[1]In the sense that the method in [7] can exactly recover that in [6] if dynamics is considered to be affine in control. In other words, the iterative method in [7] can be seen as the generalization of the latter method.

1) We propose a generic MPPI framework for autonomous navigation in cluttered 2D and 3D environments, which are inherently uncertain and partially observable. To the best of our knowledge, this point has not been reported in the literature, which opens up new directions for research.
2) We demonstrate this framework on a set of simulated quadrotor navigation tasks using RotorS simulator and Gazebo [12], assuming that: (i) there is a priori knowledge about the environment (namely, fully observable case); (ii) there is no a priori information (namely, partially observable case). In this case, the robot is building and updating the map, which represents the environment, online as it goes along. This allows the opportunity to navigate in dynamic environments.
3) To ensure a realistic simulation, our proposed framework is evaluated taking into account the modelling errors, noisy sensors, and windy environments.

This paper is organized as follows. Section II describes the quadrotor dynamics model which represents our case study for the framework validation, whereas in Section III the real-time MPPI control strategy is explained. Our proposed framework is then described in Section IV and evaluated in Section V. Finally, concluding remarks are provided in Section VI.

II. Quadrotor Dynamics Model

Considering a quadrotor vehicle model illustrated in Fig. 1, the dynamics model can be defined by assigning a fixed inertial frame $\mathcal{W}$ and body frame $\mathcal{B}$ attached to the vehicle. The origin of the body frame, $\mathcal{B}$, is located at the center of mass of the quadrotor, where x_B and y_B lie in the quadrotor plane defined by the centers of the four rotors, and z_B is perpendicular to this plane and points upward. The inertial reference frame, $\mathcal{W}$, is defined by x_W, y_W, and z_W, with z_W pointing upward. We assume that the first rotor (i.e., along the $+x_B$ axis) and the third rotor rotate counter clockwise (namely, $+1$), whilst the second (i.e., along the $+y_B$ axis) and fourth rotors rotate clockwise (namely, -1). Here, the vehicle is only subject to: (i) a gravitational acceleration g in $-z_W$ direction; (ii) the sum of the forces generated by each rotor, $F = \sum_{i=1}^{4} F_i$, acts along the $+z_B$ direction. Furthermore, the Euler angles, with ZXY transformation

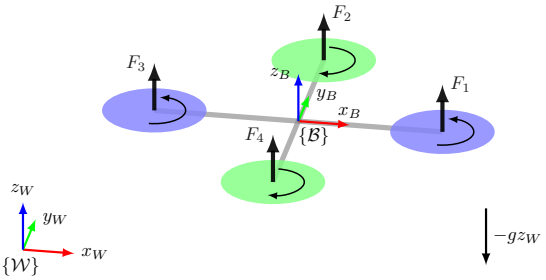

Fig. 1. Schematic of the considered quadrotor model in conjunction with the coordinate systems and forces acting on a vehicle frame.

sequence, are used to model the rotation of the quadrotor in frame $\mathcal{W}$, where the roll ϕ, pitch θ, and yaw ψ angles refer to a rotation about x_B, y_B, and z_B axis, respectively. The rotation matrix from $\mathcal{B}$ to $\mathcal{W}$ is accordingly expressed as

$$ {}^{\mathcal{W}}\mathcal{R}_{\mathcal{B}} = \begin{bmatrix} c_\psi c_\theta - s_\phi s_\psi s_\theta & -c_\phi s_\psi & c_\psi s_\theta + c_\theta s_\phi s_\psi \\ c_\theta s_\psi + c_\psi s_\phi s_\theta & c_\phi c_\psi & s_\psi s_\theta - c_\psi c_\theta s_\phi \\ -c_\phi s_\theta & s_\phi & c_\phi c_\theta \end{bmatrix}, $$

where $s_x = \sin(x)$ and $c_x = \cos(x)\ \forall\ x \in \{\phi, \theta, \psi\}$. The transformation matrix from Euler angular velocities, $\dot{\Phi}$, to body frame angular velocity, Ω, is given by

$$ T = \begin{bmatrix} c_\theta & 0 & -c_\phi s_\theta \\ 0 & 1 & s_\phi \\ s_\theta & 0 & c_\phi c_\theta \end{bmatrix}. $$

Given all considerations above and according to the well-described models in [1] and [13], the dynamics of the position ξ, linear velocity v, orientation locally defined by Euler angles Φ, and body angular rates Ω can be written as

$$ \begin{aligned} \dot{\xi} &= v, & m\dot{v} &= -mge_3 + {}^{\mathcal{W}}\mathcal{R}_{\mathcal{B}} F e_3, \\ \dot{\Phi} &= T^{-1}\Omega, & J\dot{\Omega} &= \Gamma - \Omega \times J\Omega, \end{aligned} \tag{1} $$

where m is the mass of the quadrotor and J its inertia matrix expressed in the body frame $\mathcal{B}$, $\xi = [x, y, z]^T$, $v = [v_x, v_y, v_z]^T$, $\Phi = [\phi, \theta, \psi]^T$, $\Omega = [p, q, r]^T$, $e_3 = [0, 0, 1]^T \in \mathbb{R}^3$, $F \in \mathbb{R}^+$ is the accumulated force (i.e., thrust) generated by all rotors and constitutes the first control input to the system, $\Gamma = [\tau_x, \tau_y, \tau_z]^T = [L(F_2 - F_4), L(F_3 - F_1), (M_1 - M_2 + M_3 - M_4)]^T \in \mathbb{R}^3$ is the total torque applied to the vehicle with its components expressed in $\mathcal{B}$ which represents the second control input, and $L \in \mathbb{R}^+$ is the distance from the center of the vehicle to the axis of rotation of each rotor. In this paper, the state of the system is defined as $\mathbf{x} = [x, y, z, \phi, \theta, \psi, v_x, v_y, v_z, p, q, r]^T \in \mathbb{R}^{12}$. Each rotor produces a vertical force, F_i, and moment, M_i, according to $F_i = k_F \omega_i^2$ and $M_i = k_M \omega_i^2$, where ω_i is the angular velocity of the i^{th} rotor, k_F is the rotor force constant, and k_M is the rotor moment constant. Accordingly, the mapping between the control inputs, namely F and Γ, and the system's input, i.e., ω_i, in order to control the quadrotor, can be expressed as

$$ \begin{bmatrix} F \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}. \tag{2} $$

III. MPPI Control Strategy

The MPPI controller is a stochastic Model Predictive Control (MPC) method, which can be applied to non-linear dynamics and non-convex cost objectives. So, it is a sampling-based and derivative-free approach. The key idea of MPPI is to sample thousands of trajectories, based on Monte-Carlo simulation, in real-time from the system dynamics. Each trajectory is then evaluated according to a predefined cost function. Consequently, the optimal control sequence is

updated over all trajectories. This can be easily done, in real-time, by taking advantage of the parallel nature of sampling and using a Graphics Processing Unit (GPU).

Let assume that the discrete-time stochastic dynamical system has a form of

$$\mathbf{x}_{t+1} = f\left(\mathbf{x}_t, \mathbf{u}_t + \delta\mathbf{u}_t\right), \tag{3}$$

where $\mathbf{x}_t \in \mathbb{R}^n$ is the state vector of the system at time t, $\mathbf{u}_t \in \mathbb{R}^m$ denotes a control input for the system, and $\delta\mathbf{u}_t$ is a zero-mean Gaussian noise vector with a variance of $\Sigma_{\mathbf{u}}$, i.e., $\delta\mathbf{u}_t \sim \mathcal{N}(\mathbf{0}, \Sigma_{\mathbf{u}})$, which represents the control input updates. Given a finite time-horizon $t \in \{0, 1, 2, \cdots, T-1\}$, the objective of the stochastic optimal control problem is to find a control sequence, $\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, \ldots \mathbf{u}_{T-1}) \in \mathbb{R}^{m\times T}$, which minimizes the expectations over all generated trajectories taken with respect to (3), i.e., $J = \min_{\mathbf{u}} \mathbb{E}\left[S\left(\tau\right)\right]$, where $S\left(\tau\right) \in \mathbb{R}$ is the state-dependent cost-to-go of a trajectory $\tau = \{\mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \cdots, \mathbf{u}_{T-1}, \mathbf{x}_T\}$. Accordingly, the optimal problem can be formulated as

$$J = \min_{\mathbf{u}} \mathbb{E}\left[\phi\left(\mathbf{x}_T\right) + \sum_{t=0}^{T-1}\left(q\left(\mathbf{x}_t\right) + \frac{1}{2}\mathbf{u}_t^T R \mathbf{u}_t\right)\right], \tag{4}$$

where $\phi\left(\mathbf{x}_T\right)$, $q\left(\mathbf{x}_t\right)$, and $R \in \mathbb{R}^{m\times m}$ are a final terminal cost, a state-dependent running cost, and a positive definite control weight matrix, respectively. To solve this optimization problem, we consider the iterative update law derived in [6], in which MPPI algorithm updates the control sequence, from t onward, as

$$\mathbf{u}_t \leftarrow \mathbf{u}_t + \frac{\sum_{k=1}^{K} \exp\left(-(1/\lambda)\tilde{S}\left(\tau_{t,k}\right)\right)\delta\mathbf{u}_{t,k}}{\sum_{k=1}^{K} \exp\left(-(1/\lambda)\tilde{S}\left(\tau_{t,k}\right)\right)}, \tag{5}$$

where K is the number of random samples (namely, roll-outs), $\lambda \in \mathbb{R}^+$ is a hyper-parameter so-called the inverse temperature, and $\tilde{S}\left(\tau_{t,k}\right) = \phi\left(\mathbf{x}_T\right) + \sum_{t=0}^{T-1} \tilde{q}\left(\mathbf{x}_t, \mathbf{u}_t, \delta\mathbf{u}_t\right)$ is the modified cost-to-go of the k^{th} rollout from time t onward.

In this work, the modified running cost $\tilde{q}\left(\mathbf{x}, \mathbf{u}, \delta\mathbf{u}\right)$ is defined as

$$\tilde{q} = q(\mathbf{x}) + \frac{1}{2}\mathbf{u}^T R\mathbf{u} + \frac{1-\nu^{-1}}{2}\delta\mathbf{u}^T R\delta\mathbf{u} + \mathbf{u}^T R\delta\mathbf{u}, \tag{6}$$

where $\nu \in \mathbb{R}^+$ refers to the exploration noise which determines how aggressively the state-space is explored. It is noteworthy that the low values of ν result in the rejection of many sampled trajectories because their cost is too high, while too large values result in that the controller produces control inputs with significant chatter.

The real-time control cycle of MPPI algorithm is described in Algorithm 1 with more detail. At each time-step Δt, the system current state is estimated, and a $K \times T$ random control variations are generated on a GPU using CUDA's random number generation library (lines $2:3$). Then, based on the parallel nature of sampling, all trajectory samples are executed individually in parallel. For each trajectory, the dynamics are predicted forward and its expected cost is computed (lines $5:12$), bearing in mind that the cost of each trajectory is zero-initialized (line 4). The control sequence is then updated (lines $14:16$), taking into account the minimum sampled cost $\tilde{S}_{\min}$ (line 13). Due to the stochastic nature of the sampling procedure which leads to significant chattering in the resulting control, the control sequence is then smoothed using a Savitzky-Galoy (SG) filter (line 17). Finally, the first control is executed (line 18), while the remaining sequence of length $T-1$ is slid down to be used at next time-step Δt (lines $19:22$).

Algorithm 1 Real-Time MPPI Control Scheme [6]

Given:
K, T: Number of rollouts (samples) & timesteps
$(\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{T-1}) \equiv \mathbf{u}$: Initial control sequence
$f, \Delta t$: Dynamics & time-step size
$\phi, q, \lambda, \nu, \Sigma_{\mathbf{u}}, R$: Cost functions/hyper-parameters
SGF: Savitzky-Galoy (SG) convolutional filter

1: **while** task not completed **do**
2: $\mathbf{x}_0 \leftarrow$ StateEstimator(), $\mathbf{x}_0 \in \mathbb{R}^n$
3: $\delta\mathbf{u} \leftarrow$ RandomNoiseGenerator(), $\delta\mathbf{u} \in \mathbb{R}^{K\times T}$
4: $\tilde{S}\left(\tau_k\right) \leftarrow$ TrajectoryCostInitializer(), $\tilde{S}\left(\tau_k\right) \in \mathbb{R}^K$
5: **for** $k \leftarrow 0$ **to** $K-1$ **do**
6: $\mathbf{x} \leftarrow \mathbf{x}_0$
7: **for** $t \leftarrow 0$ **to** $T-1$ **do**
8: $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + f\left(\mathbf{x}_t, \mathbf{u}_t + \delta\mathbf{u}_{t,k}\right)\Delta t$
9: $\tilde{S}\left(\tau_{t+1,k}\right) \leftarrow \tilde{S}\left(\tau_{t,k}\right) + \tilde{q}$
10: **end for**
11: $\tilde{S}\left(\tau_k\right) \leftarrow \tilde{S}\left(\tau_{t+1,k}\right) + \phi\left(\mathbf{x}_T\right), \forall t = T-1$
12: **end for**
13: $\tilde{S}_{\min} \leftarrow \min_k[\tilde{S}\left(\tau_k\right)]$
14: **for** $t \leftarrow 0$ **to** $T-1$ **do**
15: $\mathbf{u}_t \leftarrow \mathbf{u}_t + \frac{\sum_{k=0}^{K} \exp\left(\frac{1}{\lambda}\left[\tilde{S}(\tau_{t,k}) - \tilde{S}_{\min}\right]\right)\delta\mathbf{u}_{t,k}}{\sum_{k=0}^{K} \exp\left(\frac{1}{\lambda}\left[\tilde{S}(\tau_{t,k}) - \tilde{S}_{\min}\right]\right)}$
16: **end for**
17: $\mathbf{u} \leftarrow$ SGF($\mathbf{u}$)
18: $\mathbf{u}_0 \leftarrow$ SendToActuators($\mathbf{u}$)
19: **for** $t \leftarrow 1$ **to** $T-1$ **do**
20: $\mathbf{u}_{t-1} \leftarrow \mathbf{u}_t$
21: **end for**
22: $\mathbf{u}_{T-1} \leftarrow$ ControlSequenceInitializer($\mathbf{u}_{T-1}$)
23: Check for task completion
24: **end while**

IV. Generic MPPI Framework

In this section, we present a generic and elegant MPPI framework, as illustrated in Fig. 2, in order to not only navigate autonomously in previously unseen 2D or 3D environments while avoiding collisions with obstacles but also to explore and map them. Moreover, we describe how the environment is represented to be used by MPPI for partially and fully observable navigation tasks. In Fig. 2, we present the block diagram of our proposed control framework integrated into the Robot Operating System (ROS). The individual components of our framework are described in detail in the following sections.

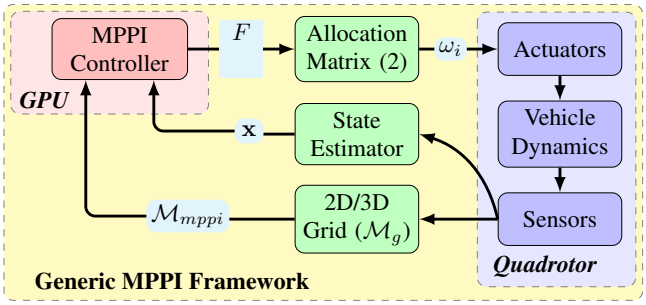

Fig. 2. The global architecture of our proposed framework.

A. Environment Representation

To clarify how MPPI can be used for 2D or 3D navigation in unseen environments, we assume that the environment is discretized into a 2D or 3D grid $\mathcal{M}_g$ (see Fig. 2), where each cell is labeled as free, occupied, or unknown, i.e., $\mathcal{M}_g = \mathcal{M}_{\text{free}} \cup \mathcal{M}_{\text{occ}} \cup \mathcal{M}_{\text{unk}}$. In practice, this labeling can be acquired from depth sensors using OctoMap [14]. As the 3D grid is a grid of cubic volumes of equal size called voxels, we can represent the environment by a set of layers ℓ_N along z_W direction, where $\ell_N = (\frac{\ell_z}{r})$, ℓ_z refers to the real environment's height, and r is the voxel size. Accordingly, each layer represents a 2D occupancy grid, producing in-total ℓ_N 2D grids for a given environment, as illustrated in Fig. 3. Since this work is mainly concerned with the control framework, the perception problem is not presently covered. Thus, the perception is here imitated by the so-called 2D or 3D mask $\mathcal{F}_{fov}$, which represents the sensor's field of view (FoV) as the robot cannot normally perceive the entire environment. For the sake of simplicity, we assume that the

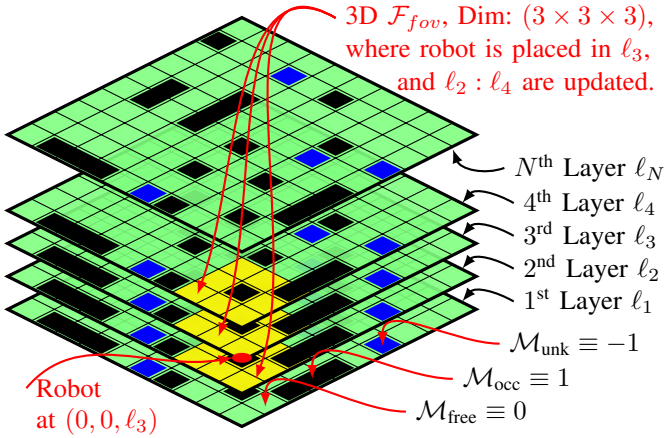

Fig. 3. 3D environment representation of a generic MPPI.

3D $\mathcal{F}_{fov}$ has dimensions $f_x \times f_y \times f_z$ and an orientation f_θ, where f_x and f_y represent the number of cells in the 2D grid along x_W and y_W axes. While f_z represents the number of layers along z_W, assuming that the robot's vertical FoV f_N is between $[(int(\frac{-f_z}{2}) + \ell_n) : (int(\frac{f_z}{2}) + \ell_n)]$ where the robot is located in layer ℓ_n (see Fig. 3). Note that the 3D $\mathcal{F}_{fov}$ is equivalent to f_N 2D $\mathcal{F}_{fov}$.

B. Fully/Partially Observable 2D/3D Navigation Tasks

Let assume that $\mathcal{M}_g$ represents the global map representation of a given environment. This map is initialized using a priori knowledge about the environment. Let $\mathcal{M}_{mppi}$ be the local map of MPPI, which has the same size as $\mathcal{M}_g$ and each cell is initialized with -1 referring to unknown cells. This local map is continuously updated using an onboard sensing system, as depicted in Fig. 2.

1) Fully Observable Case: In a fully observable case, MPPI is directly provided with a global map $\mathcal{M}_g$ to compute the trajectories' cost for avoiding obstacles, taking into account the current robot position obtained from a state estimator. For a 2D navigation task, a 2D floor grid is sufficient; only the first layer, ℓ_1, of 3D voxel-grid will be accessed by MPPI. While, for the 3D case, the 3D voxel-grid is required for performing collision-free navigation. Clearly, given the robot's position, particularly its z-component, the corresponding layer, ℓ_n, to z-component is used for evaluating the cost-to-go of each sampled trajectory (see Fig. 3). At the moment, the main limitation of the fully observable case is that our framework is only able to handle static environments; this is an important motivation for defining a partially observable navigation task that is prevalent in robotics applications.

2) Partially Observable Case: In this case, the robot is located in a previously unseen and dynamic environment and must navigate to: (i) a predefined goal, (ii) or explore and map that area. As the environment is unknown and $\mathcal{M}_g$ is not directly accessible by MPPI, the robot must perceive its environment (in our case, through the predefined mask $\mathcal{F}_{fov}$) and react appropriately. Here, the MPPI map $\mathcal{M}_{mppi}$, which is initialized with -1, is fed directly into the control framework as a local map for avoiding obstacles. This map is continuously updated as the robot moves around. For the sake of clarity, let us consider the map in Fig. 3 as $\mathcal{M}_g$, where the robot is located in $(0, 0, \ell_3)$ and $\mathcal{F}_{fov}$ has dimensions of $3 \times 3 \times 3$. As a consequence, based on the intersection between $\mathcal{M}_g$ and $\mathcal{F}_{fov}$, the layers from ℓ_2 to ℓ_4 in $\mathcal{M}_{mppi}$ are updated, while other layers have remained constant.

Algorithm 2 Real-Time Generic MPPI Framework

Given:
$\mathcal{M}_g, \mathcal{M}_{mppi}$: Global and local map of MPPI
$\mathcal{F}_{fov}$: Sensor's FoV and its parameters

while task not completed **do**
  $\mathbf{x}_0 \leftarrow$ StateEstimator(), $\mathbf{x}_0 \in \mathbb{R}^n$
  $\mathcal{M}_{mppi} \leftarrow$ MapUpdate($\mathcal{M}_g \cap \mathcal{F}_{fov}$)
  $\mathbf{u}_0 \leftarrow$ MPPIController($\mathcal{M}_{mppi}$)
  Check for task completion
end while

Generally speaking, the real-time implementation of our proposed generic MPPI framework for 2D or 3D navigation in cluttered environments is better described in Algorithm 2,

which employs the MPPI control scheme described above. At each time-step Δt, the current state is estimated (line 2). The local map of MPPI $\mathcal{M}_{mppi}$ is then updated accordingly, given the global map $\mathcal{M}_g$ (line 3). As discussed previously, $\mathcal{M}_g$ in conjunction with $\mathcal{F}_{fov}$ is used for updating the local map $\mathcal{M}_{mppi}$, as the perception modules have not been considered in the current work. Thus, in practice, the robot must be equipped with an on-board sensing system, e.g. depth camera or laser scanner, with a maximum FoV $\mathcal{F}_{fov}$. This currently sensed data is used to obtain a 2D or 3D occupancy map, which is continuously updated, i.e., $\mathcal{M}_{mppi}$. For instance, a quadrotor-based exploration algorithm is proposed in [15] to build a real-time 3D map. Finally, this map enables the controller to find the optimal control $\mathbf{u}_0$ to be executed, resulting in collision-free navigation (line 4).

V. Simulation Details and Results

In this section, we describe how we evaluate our approach in terms of simulation scenarios and performance metrics. Moreover, we conduct realistic simulations to evaluate and demonstrate the performance of the proposed framework.

A. Simulation Setup, Scenarios, and Metrics

1) Simulation Setup: In order to evaluate the performance of our proposed MPPI framework in a previously unseen and cluttered environment, simulation studies have been performed using RotorS simulator and Gazebo [12]. The parameters of the real simulated quadrotor (namely, Humminbird quadrotor) are tabulated in Table I. To ensure a realistic RotorS-based simulation, all navigation tasks are carried out by (i) using noisy sensors such as GPS and IMU, (ii) adding external disturbances such as continuous wind with changing speed and direction, as proposed in [16], and (iii) considering $\pm 10\%$ of modelling errors in the real values of mass m and inertia J of the prediction model given in (1). The MPPI controller has a time horizon t_p of $3\,\mathrm{s}$, a control frequency of $50\,\mathrm{Hz}$, and generates 2700 samples each time-step Δt. The rest of its hyper-parameters are also listed in Table I, where the 2.5 value in $\Sigma_{\mathbf{u}}$ represents the noise in the thrust input F and $R = \lambda \Sigma_{\mathbf{u}}^{-1}$. For the SG filter, we set the length of filter window and order of the polynomial function to 51 and 3, respectively. The real-time execution of MPPI is performed on a GeForce GTX 1080 Ti, where all algorithms are written in Python and have been implemented using ROS.

TABLE I
Parameters of Quadrotor and MPPI

Parameter	Value	Parameter	Value
m [kg]	0.716	t_p [s]	3
L [m]	0.17	T	150
g [$\mathrm{m\,s^{-2}}$]	9.81	K	2700
k_F [$\mathrm{N\,rpm^{-2}}$]	8.55×10^{-6}	λ	0.02
k_M [$\mathrm{N\,rpm^{-2}}$]	1.6×10^{-2}	ν	1000
J [$\mathrm{kg\,m^2}$]	$\mathrm{Diag}\left(7 \times 10^{-3}, 7 \times 10^{-3}, 12 \times 10^{-3}\right)$		
$\Sigma_{\mathbf{u}}$	$\mathrm{Diag}\left(2.5, 5 \times 10^{-3}, 5 \times 10^{-3}, 5 \times 10^{-3}\right)$		
R	$\mathrm{Diag}\left(8 \times 10^{-3}, 4, 4, 4\right)$		

In our RotorS-based simulations, the full-state information, $\mathbf{x} = [x, y, z, \phi, \theta, \psi, v_x, v_y, v_z, p, q, r]^T$, is directly provided, using an *odometry* estimator based on an Extended Kalman Filter (EKF), as an input to the controller. While, since the actual control signal of the quadrotor is the angular velocity of each rotor ω_i, the controller outputs F and Γ are directly converted into ω_i using (2), to be sent to the quadrotor as *Actuators* message. In summary, the closed-loop of MPPI in ROS, at each Δt, can be summarized as: (i) MPPI first receives the *odometry* message and the updated map $\mathcal{M}_{mppi}$ obtained from the on-board sensor; (ii) the control action is then computed and published. As mentioned before, the whole process of our proposed framework is summarized in Fig. 2.

Since we are interested in goal-oriented quadrotor navigation tasks in cluttered environments, the state-dependent cost function is defined as

$$q(\mathbf{x}) = (\mathbf{x} - \mathbf{x}^{\text{des}})^{\mathrm{T}} Q (\mathbf{x} - \mathbf{x}^{\text{des}}) + 10^8 C_1 + 10^5 C_2,$$

where:

$$C_1 = \{\mathbf{x} : \ (z < 0) \ \textbf{or} \ (\mathcal{M}_{mppi}[x, y, z] = 1)\},$$

$$C_2 = \{\mathbf{x} : \ (\|v\| > v_{\max}) \ \textbf{or} \ (((|\cos\phi|) \ \textbf{or} \ (|\cos\theta|)) < 0.1) \ \textbf{or} \ (z > 8.5)\},$$

$$\mathbf{x}^{\text{des}} = (x^{\text{des}}, y^{\text{des}}, z^{\text{des}}, 0, 0, \psi^{\text{des}}, \mathrm{zeros}(1, 6)),$$

$$Q = \begin{cases} \mathrm{Diag}(2.5, 2.5, 5, 1, 1, 50, \mathrm{zeros}(1, 6)), \ \forall v_{\max} \leq 1.5 \frac{\mathrm{m}}{\mathrm{s}}, \\ \mathrm{Diag}(5, 5, 15, 30, 30, 50, \mathrm{zeros}(1, 6)), \ \text{Otherwise.} \end{cases}$$

The first term C_1 indicates the collision with ground or obstacles, while C_2 prevents the quadrotor from (i) going too fast, (ii) using too aggressive roll and pitch angles, or (iii) colliding with the ceiling, where C_1 and C_2 are boolean variables. $\mathbf{x}^{\text{des}}$ refers to the desired position to be reached and its orientation, while Q is a weighing matrix.

2) Simulation Scenarios: Two different scenarios are considered for evaluating the proposed framework: *2D scenario* and *3D scenario*. The *2D scenario* refers to a $40 \times 40 \times 8.5\,\mathrm{m}$ windy forest of cylindrical obstacles placed in a 2D grid pattern, where each cylinder has a radius of $0.16\,\mathrm{m}$ with equal spacing of $4\,\mathrm{m}$ apart. The *3D scenario* refers to the same environment described in the *2D scenario*, while we have added two horizontal layers of cylinders at $z = 3\,\mathrm{m}$ and $6\,\mathrm{m}$ with the same spacing each, as shown in Fig. 4. The voxel size r is set to $0.2\,\mathrm{m}$, to ensure high accuracy and to meet the reality when a 3D occupancy grid is involved. As a result, the 3D voxel grid $\mathcal{M}_g$ has 43 layers, each layer represents a 40×40 2D grid.[2] The *3D scenario* is used for performing fully and partially observable 3D navigation tasks, while the former is used for the 2D tasks. For both scenarios, two different cases are considered: (i) Fully Observable Case (*FOC*), in which an a priori knowledge about the environment is used to initialize the global map $\mathcal{M}_g$; in *FOC*, $\mathcal{M}_{mppi}$ is exactly $\mathcal{M}_g$; and (ii) Partially Observable Case (*POC*), where it is assumed that there is no a priori information about the environment; for this reason, the local

[2]The cell size of 2D grid is $1\,\mathrm{m}$ (i.e., 1:1 scale), since all obstacles, in our case, are placed in a 2D grid pattern with integer numbers.

map $\mathcal{M}_{mppi}$ is discovered and built online as the robot moves around.

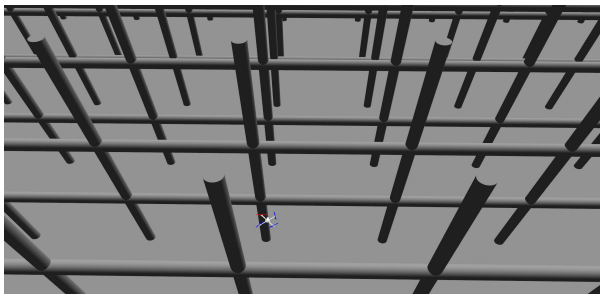

Fig. 4. The 3D cluttered environment used for 3D tasks.

3) Performance Metrics: We define two metrics for evaluation. First, the performance of MPPI in *POC* is compared to its performance in *FOC*, considering the latter case as a baseline. To achieve a fair comparison, in all simulations, the robot navigates to the same specified goals with a maximum velocity $v_{\max}$ of $1.5\,\mathrm{m\,s^{-1}}$. The predefined goals (in $[\mathrm{m}]$) in order are: $G_1 = (23, 38, 1.5), G_2 = (40, 23, 4), G_3 = (22, 0, 8), G_4 = (0, 22, 5)$, then the UAV will land. While, at each Δt, the desired yaw angle ψ^{des} is updated, letting the front camera points towards the next goal. For both cases (*FOC*/*POC*), we use a number of indicators, as a second metric, to describe the general performance such as the number of collisions N_{col}, task completion percentage t_{comp}, average completion time t_{av}, average flying distance d_{av}, average flying speed v_{av}, and average energy consumption E_{av} (for more details, we refer to [16]). In all simulations, the task is considered to be terminated if the quadrotor reached the given goals (i.e., $t_{\text{comp}} = 100\%$) or crashed into an obstacle (i.e., $N_{\text{col}} = 1$).

B. Simulation Results

The performance of our proposed framework is validated for both fully and partially observable 2D/3D quadrotor navigation tasks, considering the predefined goals. In all partially observable tasks, we set $\mathcal{F}_{fov}$ to $5 \times 5 \times 3$ m, while f_θ represents the angle between the current and next goal. To test whether the quadrotor is able to navigate successfully through the cluttered environment, we performed 5 trials for both *FOC* and *POC*. The reader is invited to watch the whole simulation results at https://bit.ly/2PAbESO.

1) 2D Navigation Results: Figure 5 shows an example of a final trajectory generated by MPPI in the case of (i) using a 2D global map $\mathcal{M}_g$ for *FOC*, or (ii) using only a 2D local map $\mathcal{M}_{mppi}$ for *POC*. The 2D-floor $\mathcal{M}_g$ (with grey circles) and its updated $\mathcal{M}_{mppi}$ (with blue circles, representing the obstacles within the robot's $\mathcal{F}_{fov}$), for given goals, are shown in Fig. 6, including the generated trajectories in both cases and the robot's FoV $\mathcal{F}_{fov}$. In both 2D *FOC* and *POC*, it can be seen that MPPI is able to safely navigate through the windy obstacle field, in spite of the presence of modelling errors and measurements noise.

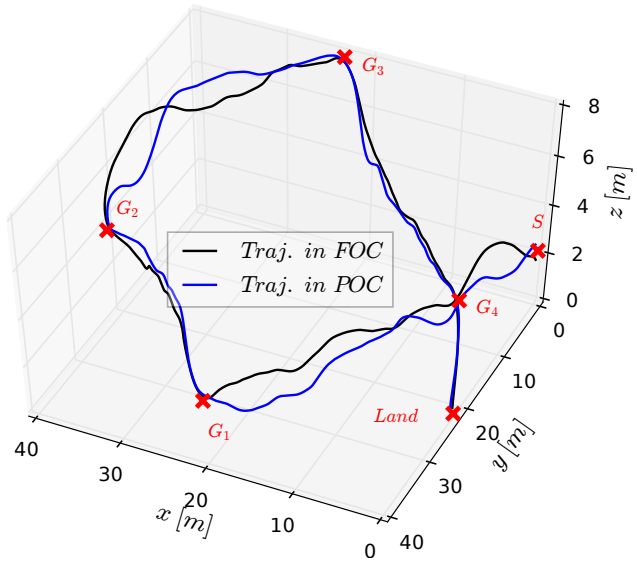

Fig. 5. Comparison of 3D trajectories in a *2D scenario*.

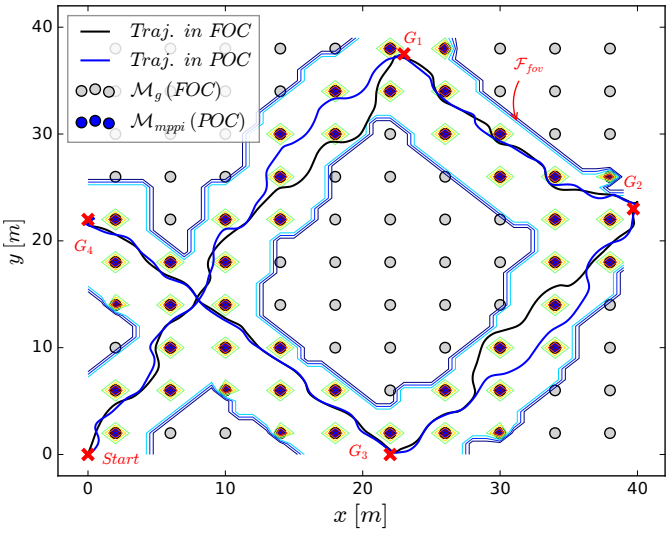

Fig. 6. The $40\,\mathrm{m}$ obstacle field map, i.e., 2D $\mathcal{M}_g$, and its updated 2D $\mathcal{M}_{mppi}$ map using a 5×5 2D $\mathcal{F}_{fov}$. Two lines represent the trajectories generated by MPPI in both cases, i.e., *FOC* and *POC*.

2) 3D Navigation Results: Another example of successfully generated trajectories for fully and partially observable 3D navigation tasks in a *3D scenario* environment is depicted in Fig. 7. In this scenario, since $\mathcal{M}_{mppi}$ consists of 43 2D grids, it is very difficult to visualize clearly in this paper the updates over all layers and insure that the robot performs collision-free navigation. So, first, we show only the 11^{th} layer ℓ_{11}, which has been mainly updated while the robot was moving towards G_1 and while it was landing after task completion, as an example of how the 3D $\mathcal{M}_{mppi}$ is updated (see Fig. 8). Second, it is highly recommended to use the indicators described previously, especially for 3D cases.

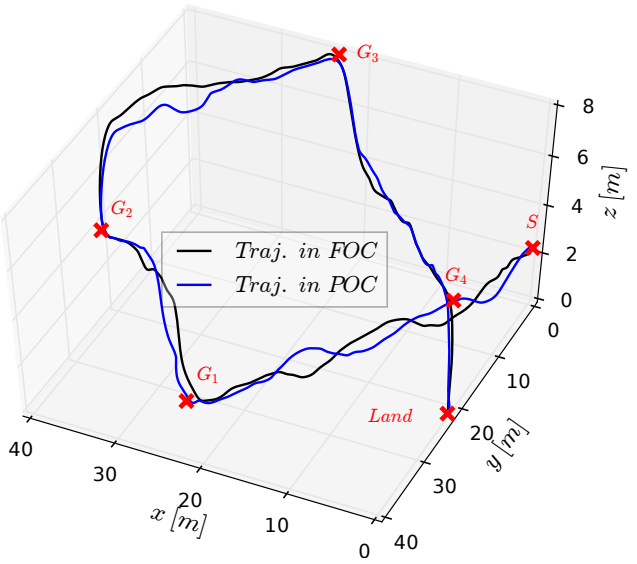

Fig. 7. Comparison of 3D trajectories in a *3D scenario*.

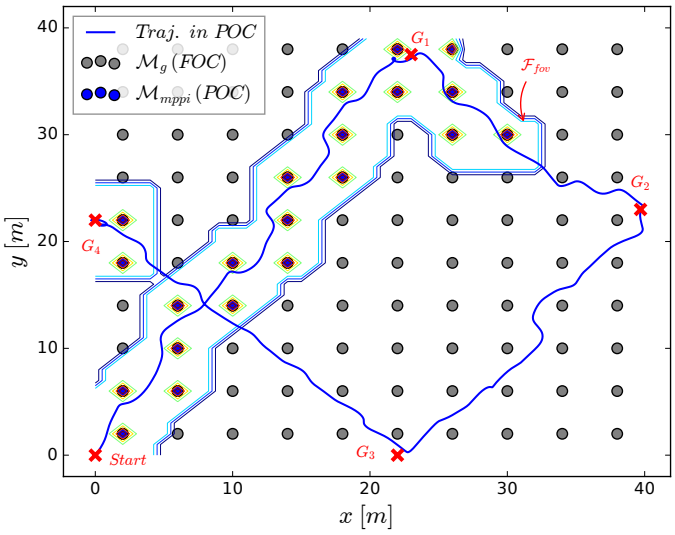

Fig. 8. The update of the 11[th] layer ℓ_{11} in the 3D $\mathcal{M}_{mppi}$, showing the generated trajectory in *POC*.

3) Overall Performance: Table II shows the performance statistics for the five trials in 2D and 3D scenarios, considering the parameters of the controller tabulated in Table I. In all trials (i.e., 20 trials in total), we can observe that the quadrotor navigates autonomously (i.e., $N_{\text{col}} = 0$) while avoiding obstacles with an average flying speed v_{av} of about $1.2\,\text{m}\,\text{s}^{-1}$, which is closer to the maximum specified velocity, i.e., $v_{\text{max}} = 1.5\,\text{m}\,\text{s}^{-1}$, regardless of the limited FoV of the sensor in both 2D and 3D scenarios. Note that these 20 trials are equivalent to more than $3\,\text{km}$ of autonomous navigation. Moreover, it can be seen that the average execution time of MPPI per iteration, t_{mppi}, in the 3D scenario is approximately equal to that in the 2D scenario (with very low standard deviation values), showing the superiority of the proposed framework even with 3D-voxel grids and its applicability to be used for 3D navigation tasks. In fact, this is not surprising as the 2D- or 3D-voxel grids are stored and directly used by MPPI on the GPU. For other indicators, we can observe that the performance of MPPI in *POC* cases is slightly different than its performance in *FOC* cases, although we have set the same parameters for all simulations. The reason behind this difference is that the control variations $\delta\mathbf{u}$ are generated randomly, each time-step Δt, on the GPU. The number of timesteps T (which is related to the time horizon t_p and control frequency) and the exploration variance ν (which is related to the number of sampled trajectories K) play an important role in determining the behavior of MPPI. Therefore, we tested six different tuning cases, where different values of T and ν have been considered, as shown in Table III. In the first three tuning cases, we study the influence of changing T where ν remains constant (i.e., $\nu = 1000$ as defined before in Table I). While the effect of changing ν is studied in the

TABLE II

GENERAL PERFORMANCE OF MPPI IN 2D AND 3D SCENARIOS FOR BOTH *FOC* AND *POC*

Indicators	**2D Scenario**		**3D Scenario**	
	FOC	*POC*	*FOC*	*POC*
$N_{\text{col}}\,(t_{\text{comp}}[\%])$	0 (100%)	0 (100%)	0 (100%)	0 (100%)
t_{av} [s]	122.1 ± 1.2	124.5 ± 3.0	128.2 ± 3.0	129.5 ± 1.4
d_{av} [m]	150.3 ± 0.8	151.1 ± 1.9	152.9 ± 1.8	153.1 ± 0.87
v_{av} [$\text{m}\,\text{s}^{-1}$]	1.23 ± 0.02	1.22 ± 0.03	1.19 ± 0.02	1.18 ± 0.01
E_{av} [W h]	6.63 ± 0.07	6.76 ± 0.16	6.96 ± 0.16	7.01 ± 0.07
t_{mppi} [ms]	18.4 ± 0.51	18.8 ± 0.42	18.2 ± 0.21	18.8 ± 0.67

TABLE III

EFFECT OF CHANGING T AND ν ON THE BEHAVIOR OF THE PROPOSED CONTROLLER

Indicators	**2D Scenario**		**3D Scenario**	
	FOC	*POC*	*FOC*	*POC*
***Tuning Case 1*: $T = 75$ (i.e., $t_p = 1.5\,\text{s}$) and $\nu = 1000$**				
$N_{\text{col}}\,(t_{\text{comp}}[\%])$	1 (57.4±30)	1 (57.4±30)	1 (40.3±6)	1 (36.3±9)
***Tuning Case 2*: $T = 100$ (i.e., $t_p = 2\,\text{s}$) and $\nu = 1000$**				
$N_{\text{col}}\,(t_{\text{comp}}[\%])$	0 (100%)	0 (100%)	0 (100%)	0 (100%)
d_{av} [m]	51.7±0.79	51.7±1.1	51.1±0.16	52.1±0.68
***Tuning Case 3*: $T = 125$ (i.e., $t_p = 2.5\,\text{s}$) and $\nu = 1000$**				
$N_{\text{col}}\,(t_{\text{comp}}[\%])$	0 (100%)	0 (100%)	0 (100%)	0 (100%)
d_{av} [m]	**50.4**±0.86	**50.7**±0.25	**50.2**±0.49	**50.5**±0.36
***Tuning Case 4*: $T = 150$ (i.e., $t_p = 3\,\text{s}$) and $\nu = 300$**				
$N_{\text{col}}\,(t_{\text{comp}}[\%])$	0 (100%)	0 (100%)	0 (100%)	0 (100%)
d_{av} [m]	52.4±0.78	52.5±0.36	52.8±0.74	53.0±0.45
***Tuning Case 5*: $T = 150$ and $\nu = 500$**				
$N_{\text{col}}\,(t_{\text{comp}}[\%])$	0 (100%)	0 (100%)	0 (100%)	0 (100%)
d_{av} [m]	51.9±0.59	51.5±0.85	52.1±0.25	52.6±0.36
***Tuning Case 6*: $T = 150$ and $\nu = 800$**				
$N_{\text{col}}\,(t_{\text{comp}}[\%])$	0 (100%)	0 (100%)	0 (100%)	0 (100%)
d_{av} [m]	**50.9**±0.25	**51.0**±0.46	**51.5**±0.93	**51.9**±0.33

last three cases, where ν was varied between 300 and 800. For the sake of simplicity, each tuning case was tested by letting the quadrotor navigates only to $G_1 = (23, 38, 1.5)$ then lands. In each tuning case, we conducted 3 trials. As a consequence, the total trials for both *FOC* and *POC* cases are 72. For *Tuning Case 1*, where a short time horizon t_p is chosen, it can be clearly noticed that the MPPI controller is unable to complete the trials at a satisfactory rate, where the success rate t_{comp} varies from 36.3% to a maximum of 57.4% in both 2D *FOC* and *POC* (1/3 successful trials). For other tuning cases (namely, from case 2 to 6), we can observe that the controller performs perfectly and is able to successfully complete all trials while avoiding obstacles. We can also notice that as T and ν increase, the performance of the controller improves. Clearly, for *Tuning Case 3* and *6* where high values of T and ν are chosen, we can see that the average flying distance d_{av} (for both 2D and 3D scenarios) is the shortest (compared to other successful cases) with low standard deviation values, which means that the quadrotor is taking a more direct route leading to the goal. However, having too long time horizons increase the computational effort dramatically because each trajectory takes more time to simulate. While, if the exploration variance ν is too large, the controller produces control inputs with significant chatter. We also notice during our simulations that higher values of control weight matrix R leads to empirically slow convergence of MPPI towards the goal and fluctuated motion. For this reason, in all experiments, we set T, ν, and R to the values given in Table I, where MPPI performs very consistently for all given goals.

VI. Conclusions and Future Work

Within this work, we proposed an extension to the classical MPPI framework that enables the robot to navigate autonomously in 2D or 3D environments while avoiding collisions with obstacles. The key point of our proposed framework is to provide MPPI with a 2D or 3D grid representing the real world to perform collision-free navigation. This framework has been successfully tested on realistic simulations using quadrotor, considering both fully and partially observable cases. The current simulations illustrate the efficiency and robustness of the proposed controller for 2D and 3D navigation tasks. Although the *theoretical* stability of MPPI has not been addressed and proofed in the literature, its *practical* stability can be achieved by setting the MPPI parameters carefully as we described previously. We will explore the possible methods that may allow in the future to study the *theoretical* stability of MPPI. Our future work will also include the implementation of the framework in practical applications.

References

[1] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012.

[2] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, Nov. 2015.

[3] K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, *et al.*, "Fast, autonomous flight in GPS-denied and cluttered environments," *Journal of Field Robotics*, vol. 35, no. 1, pp. 101–120, 2018.

[4] T. Baca, D. Hert, G. Loianno, M. Saska, and V. Kumar, "Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Madrid, Spain, Oct. 2018, pp. 6753–6760.

[5] M. Ryll, J. Ware, J. Carter, and N. Roy, "Efficient trajectory planning for high speed flight in unknown environments," in *International Conference on Robotics and Automation*, Montreal, Canada, May 2019, pp. 732–738.

[6] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344–357, 2017.

[7] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Information-theoretic model predictive control: Theory and applications to autonomous driving," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1603–1622, 2018.

[8] R. Kusumoto, L. Palmieri, M. Spies, A. Csiszar, and K. O. Arras, "Informed information theoretic model predictive control," in *International Conference on Robotics and Automation*, Montreal, Canada, May 2019, pp. 2047–2053.

[9] M. Okada and T. Taniguchi, "Acceleration of gradient-based path integral method for efficient optimal and inverse optimal control," in *IEEE International Conference on Robotics and Automation*, Brisbane, Australia, Sept. 2018, pp. 3013–3020.

[10] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Aggressive deep driving: Combining convolutional neural networks and model predictive control," in *Conference on Robot Learning*, 2017, pp. 133–142.

[11] A. Buyval, A. Gabdullin, K. Sozykin, and A. Klimchik, "Model predictive path integral control for car driving with autogenerated cost map based on prior map and camera image," in *2019 IEEE Intelligent Transportation Systems Conference*. IEEE, 2019, pp. 2109–2114.

[12] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Cham: Springer International Publishing, 2016, ch. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625.

[13] J.-M. Kai, G. Allibert, M.-D. Hua, and T. Hamel, "Nonlinear feedback control of quadrotors exploiting first-order drag effects," in *IFAC World Congress*, Toulouse, France, July 2017, pp. 8189–8195.

[14] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.

[15] T. Cieslewski, E. Kaufmann, and D. Scaramuzza, "Rapid exploration with multi-rotors: A frontier selection method for high speed flight," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vancouver, Canada, Sept. 2017, pp. 2135–2142.

[16] T. Tezenas Du Montcel, A. Negre, J.-E. Gomez-Balderas, and N. Marchand, "BOARR: A benchmark for quadrotor obstacle avoidance based on ROS and RotorS," in *French national conference on ROS (ROSCon Fr)*, Paris, France, May 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02142571