π-MPPI: A Projection-based Model Predictive Path Integral Scheme for Smooth Optimal Control of Fixed-Wing Aerial Vehicles

Edvin Martin Andrejev[1,2], Amith Manoharan[1,2], Karl-Eerik Unt[2], Arun Kumar Singh[1,2]

***Abstract*— Model Predictive Path Integral (MPPI) is a popular sampling-based Model Predictive Control (MPC) algorithm for non-linear systems. It formulates trajectory optimization in terms of sampling control sequences and combining them in a weighted average manner. In spite of its versatility, a key persisting gap in MPPI stems from the non-smoothness in the resulting optimal control sequence. The jerky control profile is particularly detrimental to systems like fixed-wing aerial vehicles (FWVs) as it leads to oscillations in the control surfaces. Existing works address the smoothness issue by penalizing control bounds and their derivatives in the cost function. However, this introduces the additional challenge of appropriately tuning the cost functions. This paper presents a fundamentally different approach. We propose adding a projection filter π that takes in nominal control sequence samples and corrects them minimally to satisfy bounds on control magnitude and their higher-order derivatives. We then apply the MPPI averaging over the filtered control samples and, hence, call our approach π-MPPI. π-MPPI improves the state-of-the-art in following respects. First, it provides a simple-to-use template to ensure arbitrary smoothness in the control sequence. Second, we apply π-MPPI to FWVs and show that it is easier to tune than the baseline variant and thus leads to a more robust performance.**

I. Introduction

Model Predictive Path Integral (MPPI) [1] is a popular gradient-free, sampling-based approach for computing optimal trajectories. The core idea of MPPI is to sample control inputs from a Gaussian distribution and then approximate the optimal control as a weighted combination of the controls. The weights are determined by evaluating the cost on the forward rollouts (trajectories) of the system, as shown in Fig. 1. Despite its versatility, it has one key limitation: the optimal controls resulting from MPPI are highly jerky and require post hoc smoothing. This, in turn, prevents its application to systems like fixed-wing aerial vehicles (FWV), where jerky control can lead to sharp oscillations in the control surfaces that can destabilize the vehicle.

Existing works address the non-smoothness issue by incorporating penalties on control magnitude and their derivatives in the cost function [2]. This process can also benefit from changes in the sampling distribution [2] or by sampling control derivatives instead of control [3]. However, one understated aspect of such an approach is the additional complexity that it introduces with regard to changes in tuning the cost function. Moreover, the control penalties might conflict with primary costs and may result in the degradation of overall performance.

[1] are with the Institute of Technology, University of Tartu, Tartu, Estonia.
[2] are with the Estonian Aviation Academy, Tartu, Estonia.

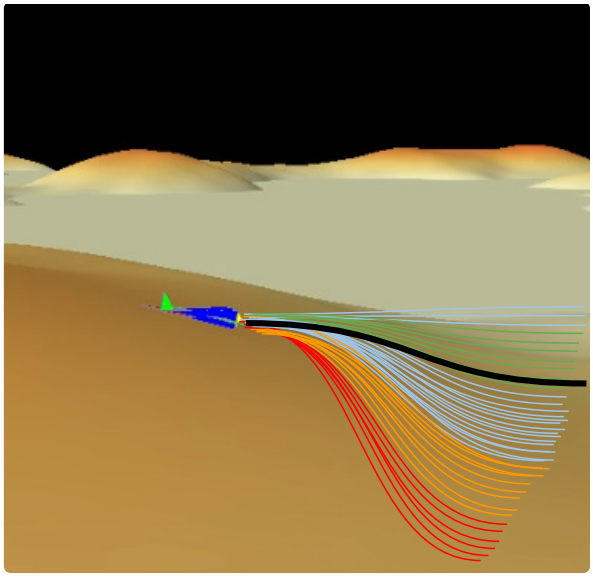


Fig. 1: Trajectory rollouts computed by the MPPI algorithm.

Algorithmic Contributions: In this paper, we present π-MPPI, a novel variant of the classical MPPI that can ensure arbitrary degrees of smoothness in the computed optimal controls. Our core idea is to embed a projection filter π that takes in samples from a baseline Gaussian distribution and modifies them in a minimal fashion to account for constraints on higher-level derivatives of the control inputs as shown in Fig. 2. We then compute the MPPI averaging over the filtered control sequences.

State-of-the-Art Performance. π-MPPI does not incorporate control penalties in the cost function. As a result, it is much easier to tune and leads to a more robust performance. We further validate this claim by considering two benchmarks involving FWVs. In the first application, the FWV is required to encircle a static target while avoiding obstacles in 3D. For the second benchmark, we consider the popular terrain-following application of FWVs, wherein it has to encircle a static target while following the terrain gradient but avoiding crashing into the surface. We show up to 50% improvement in success rate for the obstacle avoidance scenario and 30% improvement for the terrain following case. We also show that there is a significant improvement in the constraint violations for both scenarios.

II. Related Works

Recent advances in parallel computing using high-performance GPUs have opened an alternate approach of

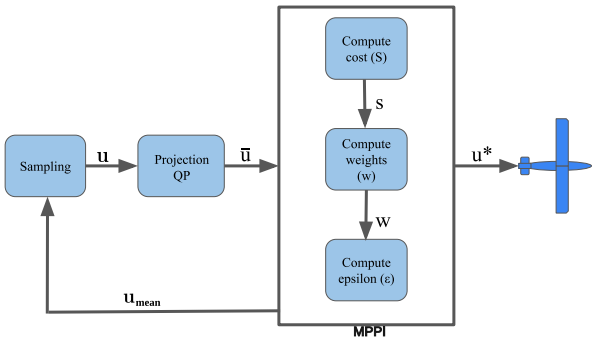


Fig. 2: Block diagram showing the proposed approach. The main difference from the baseline MPPI algorithm is introducing the projection QP block, which modifies the control samples to satisfy the constraints.

using sampling-based algorithms for the trajectory optimization problem instead of conventional optimization schemes such as model predictive control (MPC). These methods generally do not have any constraints on the cost function, and knowledge of the dynamics is often not required. The cross-entropy (CE) method is one such technique that has been widely used [4]–[8].

An overview of recent advancements in MPPI-based trajectory optimization is given in [9]. In [10], a path-following algorithm based on the MPPI is used to compute the acceleration commands for a vehicle to track a virtual target on a desired path. It was extended to consider wind disturbances in [11]. An L1-adaptive MPPI strategy for the robust control of agile multi-rotors is studied in [12]. Differentiable dynamic programming was implemented alongside the MPPI in [13] to generate aircraft trajectories over different flight envelopes. In [14], The MPPI algorithm was applied to a fixed-wing aircraft in a simulated air racing scenario.

III. Problem Formulation and Baseline MPPI

Symbols and Notation: The transpose of a vector or matrix is defined using superscript $\intercal$. Matrices are written using uppercase bold font letters. Lowercase italic letters represent scalars, and the bold font upright variants represent vectors.

A. Model of the aircraft

We use a reduced-order version of the kinematic 3D Dubins model of fixed-wing aircraft. It is a simplified model from the six degrees of freedom dynamics [15], assuming that the values of the angle of attack and the sideslip are small. The aircraft states are defined as positions along north, east, and down directions, p_n, p_e, p_d, and yaw angle, ψ.

$$\dot{p}_n = v\cos\psi\cos\theta, \tag{1}$$
$$\dot{p}_e = v\sin\psi\cos\theta, \tag{2}$$
$$\dot{p}_d = -v\sin\theta, \tag{3}$$
$$\dot{\psi} = \frac{\sin\phi}{\cos\theta}q + \frac{\cos\phi}{\cos\theta}r, \tag{4}$$

where

$$q = \frac{\dot{\theta} - r\sin\phi}{\cos\phi}, \tag{5}$$

$$r = \frac{g}{v}\sin\phi\cos\theta, \tag{6}$$

v is the velocity, ϕ and θ are the roll and pitch angles, q and r are the angular velocities about the pitch and yaw axis, and g is the acceleration due to gravity.

B. Problem Formulation

We consider a generic optimization problem of the following form:

$$\min \sum c(\mathbf{x}_k, \mathbf{u}_k), \tag{7}$$
$$\text{s.t. } \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k), \tag{8}$$
$${}^{(j)}\mathbf{u}_0 = {}^{(j)}\mathbf{u}_{\text{init}}, \tag{9}$$
$${}^{(j)}\mathbf{u}_{\min} \leq {}^{(j)}\mathbf{u}_k \leq {}^{(j)}\mathbf{u}_{\max}, \tag{10}$$

where the left superscript referes to the j^{th} order derivative of the control input $\mathbf{u}$. For FWV, we consider velocity v, roll ϕ, and pitch θ as control inputs, and their derivatives up to the second order are constrained. ${}^{(j)}\mathbf{u}_{\text{init}}$ is the initial boundary conditions for the controls and their j^{th} derivatives. The cost function c and the dynamics model f can be arbitrary nonlinear functions of the states and control.

C. Baseline MPPI Algorithm (MPPIwSGF)

We consider the algorithm presented in [16] for the baseline. Consider the discrete-time model of the system (1)-(4) represented by

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \boldsymbol{\nu}_k), \tag{11}$$

where $\mathbf{x}_k$ is the state vector and $\boldsymbol{\nu}_k = [v_k, \theta_k, \phi_k]$ is a randomly sampled input. $\boldsymbol{\nu}_k$ is not directly sampled rather, it is computed as a sum of previous mean input $\mathbf{u}_k$ and a random noise $\boldsymbol{\epsilon}_k$ drawn from a Gaussian distribution with zero mean and covariance $\boldsymbol{\Sigma_k}$ as

$$\boldsymbol{\epsilon}_k \sim \mathcal{N}(0, \boldsymbol{\Sigma_k}), \tag{12}$$
$$\boldsymbol{\nu}_k = \mathbf{u}_k + \boldsymbol{\epsilon}_k. \tag{13}$$

The discretization of the continuous-time dynamics (1)-(4) is done using the fourth-order Runge–Kutta method.

Let $s^m(\mathbf{x}_0, \boldsymbol{\nu}^m)$ be the cost associated with the m^{th} trajectory rollout, given the initial states $\mathbf{x}_0$. $\boldsymbol{\nu}^m = \left[\boldsymbol{\nu}_0^m, \boldsymbol{\nu}_1^m, \ldots, \boldsymbol{\nu}_{K-1}^m\right]$ is the sampled control sequence for the m^{th} trajectory. s^m is defined by augmenting the cost $c(\mathbf{x}_k^m, \boldsymbol{\nu}_k^m)$ with a control constraint violation cost c_p as

$$s^m = \sum_k c(\mathbf{x}_k^m, \boldsymbol{\nu}_k^m) + \beta \sum_k c_p(\boldsymbol{\nu}_k^m), \tag{14}$$

where β trades-off the effect of primary costs and control violations. The constraint penalty has the following form: for a constraint $g(\xi) \leq 0$, the $c_p = \max(0, g(\xi))$

The next step is to assign weights w^m to each trajectory as

$$w^m = \frac{1}{\eta}\exp\left(-\frac{1}{\sigma}\left(s^m + \gamma\sum_{k=0}^{K-1}\mathbf{u}_k^\intercal\boldsymbol{\Sigma}_k^{-1}\boldsymbol{\nu}_k^m - \rho\right)\right), \tag{15}$$

where

$$\eta = \sum_{m=1}^{M} \exp\left(-\frac{1}{\sigma}\left(s^m + \gamma \sum_{k=0}^{K-1} \mathbf{u}_k^{\intercal} \boldsymbol{\Sigma}_k^{-1} \boldsymbol{\nu}_k^m - \rho\right)\right), \tag{16}$$

$$\rho = \min_{m=1\ldots M} \left(s^m(\mathbf{x}_0, \boldsymbol{\nu}^m)\right), \tag{17}$$

M is the total number of sample trajectories, K is the total timesteps, and σ, γ are tunable parameters, $\gamma = \sigma(1-\alpha)$, where $0 < \alpha < 1$. When the value of σ increases, the trajectories get weighted equally. When σ decreases, the lower-cost trajectories get more weightage. The term ρ is included for numerical stability during implementation. The final step is to calculate the optimal controls for the next iteration using the weighted average of the sampled controls across all trajectories as

$$\mathbf{u}_{k,i+1} = \mathbf{u}_{k,i} + \sum_{m=1}^{M} w^m \boldsymbol{\epsilon}_k^m, \tag{18}$$

where $\mathbf{u}_{k,i}$ is the mean from the previous MPC iteration i. A Savitzky–Golay filter is used to smooth the control sequence. Hence, from here on, we refer to this method as MPPIwSGF. The main drawback of this method is the difficulty in tuning the weight β. In the next section, we propose an approach that eliminates this issue by using a projection filter π for smoothing the inputs and satisfying the control constraints. We refer to this method as the π-MPPI (Algorithm 1).

Algorithm 1: π-MPPI

Input: Dynamics f, MPPI parameters σ, γ, α, no. of timesteps K, no. of rollouts M, mean input $\mathbf{u}_{k,i}$ from the previous MPC iteration i, covariance $\boldsymbol{\Sigma}_k$, initial state $\mathbf{x}_0$, initial boundary values for the j^{th} derivatives of control ${}^{(j)}\mathbf{u}_{\text{init}}$

1 **for** $m = 1 \ldots M$ **do**
2 **for** $k = 0 \ldots K-1$ **do**
3 $\boldsymbol{\epsilon}_k^m \sim \mathcal{N}(0, \boldsymbol{\Sigma}_k)$ //Sample the noise
4 $\boldsymbol{\nu}_k^m = \mathbf{u}_{k,i} + \boldsymbol{\epsilon}_k^m$ //Random control samples
5 **end**
6 Define $\boldsymbol{\nu}^m = \left[\boldsymbol{\nu}_0^m, \boldsymbol{\nu}_1^m, \ldots, \boldsymbol{\nu}_{K-1}^m\right]$
7 Compute the projection for all the samples $\overline{\boldsymbol{\nu}}^m = \pi(\boldsymbol{\nu}^m, {}^{(j)}\mathbf{u}_{\text{init}})$
8 **end**
9 **for** $k = 0 \ldots K-1$ **do**
10 $\overline{\mathbf{u}}_{k,i}$ = Mean$\left(\left[\boldsymbol{\nu}_k^1, \boldsymbol{\nu}_k^2, \ldots, \boldsymbol{\nu}_k^M\right]\right)$ //Update mean
11 $\overline{\boldsymbol{\Sigma}}_k$ = Cov$\left(\left[\boldsymbol{\nu}_k^1, \boldsymbol{\nu}_k^2, \ldots, \boldsymbol{\nu}_k^M\right]\right)$ //Update cov.
12 **end**
13 **for** $m = 1 \ldots M$ **do**
14 **for** $k = 0 \ldots K-1$ **do**
15 $\overline{\boldsymbol{\epsilon}}_k^m = \overline{\boldsymbol{\nu}}_k^m - \overline{\mathbf{u}}_{k,i}^m$ //Update noise
16 Compute trajectory rollouts $\mathbf{x}_k^m$ using equation (11)
17 **end**
18 Compute the cost $s^m = \sum_k c(\mathbf{x}_k^m, \overline{\boldsymbol{\nu}}_k^m)$
19 Compute the weights w^m using equation (15) with the cost s^m computed from line 18, $\overline{\mathbf{u}}_{k,i}$, $\overline{\boldsymbol{\Sigma}}_k$, and $\overline{\boldsymbol{\nu}}_k^m$
20 **end**
21 **for** $k = 0 \ldots K-1$ **do**
22 Compute the weighted average control $\mathbf{u}_{k,i+1} = \overline{\mathbf{u}}_{k,i} + \sum_{m=1}^{M} w^m \overline{\boldsymbol{\epsilon}}_k^m$
23 **end**
24 Define $\mathbf{u}_{i+1} = [\mathbf{u}_{0,i+1}, \mathbf{u}_{1,i+1}, \ldots, \mathbf{u}_{K,i+1}]$
25 Compute the projection for the optimal control $\mathbf{u}_{i+1}^* = \pi(\mathbf{u}_{i+1}, {}^{(j)}\mathbf{u}_{\text{init}})$
26 $\mathbf{u}_{k,i+1} \leftarrow \mathbf{u}_{i+1}^*$ //Update the mean for next iteration

IV. Main Algorithmic Results

A. Overview of Projection based MPPI (π-MPPI)

An overview of the π-MPPI method is given in Algorithm 1. The main differences with the baseline MPPI is highlighted in blue. The first step is to sample $\boldsymbol{\nu}^m$ for K timesteps and M rollouts from a Gaussian distribution with mean $\mathbf{u}_k$ and covariance $\boldsymbol{\Sigma}_k$ as shown in lines 3-4. Then $\boldsymbol{\nu}^m$ is projected onto the constraint set using the following optimization problem to get the projected values $\overline{\boldsymbol{\nu}}^m$.

$$\min \|\boldsymbol{\nu}^m - \overline{\boldsymbol{\nu}}^m\|_2^2, \tag{19}$$

$$\text{s.t. } {}^{(j)}\overline{\boldsymbol{\nu}}_0^m = {}^{(j)}\mathbf{u}_{\text{init}}, \tag{20}$$

$${}^{(j)}\mathbf{u}_{\min} \leq {}^{(j)}\overline{\boldsymbol{\nu}}^m \leq {}^{(j)}\mathbf{u}_{\max}. \tag{21}$$

If the finite difference method is used to compute the j^{th} order derivatives, it is possible to write the optimization problem (19)-(21) as a quadratic programming problem (QP). Moreover, the computations can be performed in parallel using a GPU to accelerate the solver speed.

Since the projection operation is highly nonlinear, the mean and covariance of the control samples distribution after the projection will be different from their initial values in lines 3-4. Hence, they are updated using the projected samples as shown in lines 11-12. Similarly, the noise values are updated in line 16. The next step is to compute the trajectory rollouts using the projected samples $\overline{\boldsymbol{\nu}}_k^m$ and the kinematics function (11). Here, we don't need the control-augmented cost (14). Hence, the cost is computed using only $c(\mathbf{x}_k^m, \overline{\boldsymbol{\nu}}_k^m)$ as shown in line 19. The weights w^m are computed similarly to (15), using the mean and covariance $\overline{\mathbf{u}}_{k,i}, \overline{\boldsymbol{\Sigma}}_k$ of the projected distribution $\overline{\boldsymbol{\nu}}^m$. The weighted average of the sampled controls $\mathbf{u}_{k,i+1}$ is calculated in line 22 using the mean $\overline{\mathbf{u}}_{k,i}$, weights w^m calculated from line 19, and noise $\overline{\boldsymbol{\epsilon}}_k^m$ calculated from line 15. The weighted average controls might not strictly satisfy the control constraints. Hence, we perform the projection operation again on $\mathbf{u}_{i+1}$ to get the optimal control $\mathbf{u}_{i+1}^*$.

The main advantages over baseline MPPIwSGF can be summarized as follows

- Since the cost s^m does not have the control penalty c_p, there is no need to tune the weight β. Hence, we are able to improve the minimization of the primary cost $c(\mathbf{x}_k, \boldsymbol{\nu}_k)$.

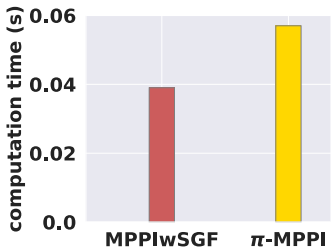


Fig. 3: Computation time required for MPPIwSGF and π-MPPI.

- As shown later in Sec. V, the controls computed using the π-MPPI are smoother compared to the MPPIwSGF.
- Even though the projection operation requires more computation, we can still achieve real-time performance by parallelizing using GPU to accelerate the process.

V. Results

We consider two applications of the proposed algorithm to show its efficacy, which are i) encircling a stationary target using a FWV while avoiding static obstacles in 3D space, ii) terrain following, where the FWV circles a target in an uneven terrain closely matching the surface altitude as shown in Fig. 4. The proposed scheme π-MPPI was compared against a baseline MPPI with Savitzky–Golay filter (MPPIwSGF) for smoothing.

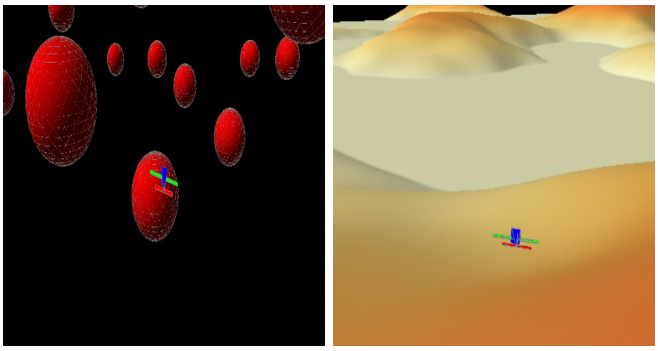
A. Implementation Details

Algorithm 1 was implemented in Python using the JAX [17] library for GPU-accelerated computing. The NN training was carried out using the [18] library. All simulations were conducted for $K = 100$ time-steps with a sampling time of $0.2\,\text{s}$ and the number of rollouts $M = 1000$ on a computer with Intel i9 CPU and Nvidea RTX 3080 GPU. The computation time requirement for both the MPPIwSGF and the π-MPPI is shown in Fig. 3. Even though the π-MPPI takes more time due to the projection operations, it is still real-time implementable with $0.057\,\text{s}$ per calculation.

B. Obstacle avoidance

In this section, we discuss the qualitative and quantitative results obtained by comparing the π-MPPI algorithm with the MPPIwSGF for the obstacle avoidance scenario. We consider a 3D space with dimensions 2000x2000x400 m filled with 100 obstacles with radii ranging from 20 m to 30 m. 250 simulations with random configurations of the obstacles and the goal (target) point were carried out for the quantitative results.

Fig. 5 shows the trajectories generated using the MPPIwSGF and the π-MPPI for an example case. It can be seen that the π-MPPI is able to encircle the target correctly with a smaller radius while avoiding the obstacles. A successful mission is defined as the case where the FWV did not collide with any obstacles along the trajectory. Fig. 6a shows the success rates for our approach and the baseline. We can clearly see the improvement in the success rate for the π-MPPI compared to the MPPIwSGF. The proposed approach also reduces the mean distance from the target as shown in Fig. 6b.



(a) Obstacle avoidance scenario (b) Terrain following scenario

Fig. 4: Two example application scenarios for the proposed scheme.

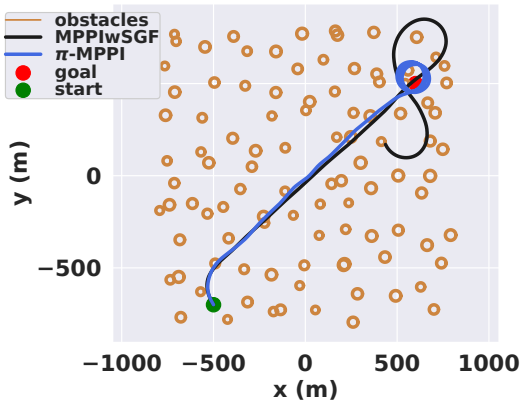
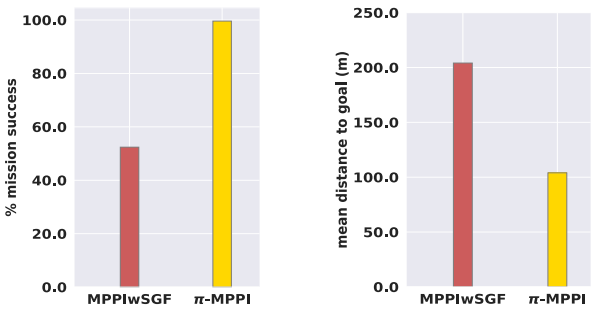


Fig. 5: Top view (2D) of the example trajectories taken by MPPIwSGF and π-MPPI for the obstacle avoidance scenario.

Another important advantage of π-MPPI is the possibility to constrain the derivatives of the control inputs to any arbitrary order. Fig. 7 shows an example case of double derivatives of v, ϕ, θ produced by π-MPPI and MPPIwSGF. It can be seen that the solution from π-MPPI is smoother compared to the MPPIwSGF with zero constraint violations. Fig. 8 shows the mean constraint violations of $\ddot{v}, \dot{\phi}, \ddot{\theta}$. The violations are near zero for the π-MPPI whereas the MPPIwSGF shows significant violations with more than $0.6\,\text{m/s}^3$



(a) Obstacle avoidance success % (b) Mean distance to goal

Fig. 6: Comparison of success rate and mean distance to the goal of MPPIwSGF and π-MPPI for the obstacle avoidance scenario.

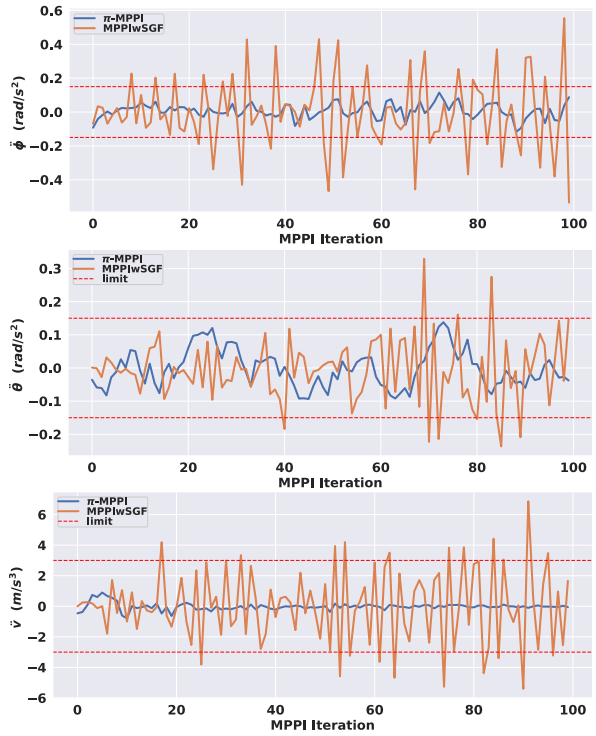


Fig. 7: Comparison of the double derivative of control inputs of MPPIwSGF and π-MPPI for an example obstacle avoidance scenario.

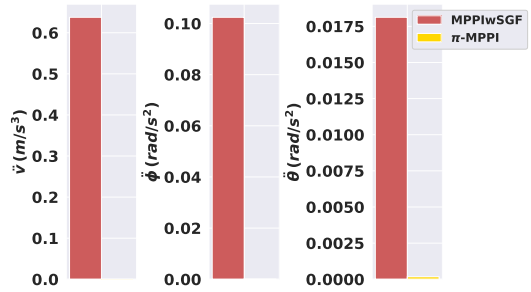


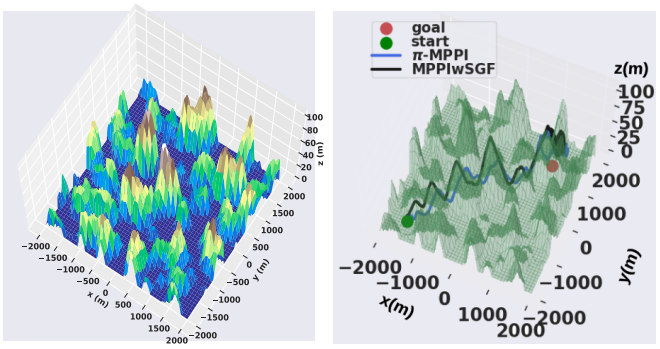
Fig. 8: Comparison of the mean constraint violations of $\dddot{v}, \ddot{\phi}, \ddot{\theta}$ of MPPIwSGF and π-MPPI for the obstacle avoidance scenario.

for the jerk and 0.10 rad/s^2 for the angular acceleration in the roll axis.

C. Terrain following

In this section, we discuss the qualitative and quantitative results obtained by comparing the π-MPPI algorithm with the MPPIwSGF for the terrain following scenario. We consider an uneven terrain with dimensions 4000x4000x100 m. 250 simulations with random configurations of the goal (target) point were carried out for the quantitative results.

Fig. 9a shows the terrain used for the simulations. The FWV trajectories generated using the MPPIwSGF and the π-MPPI for an example scenario are shown in Fig. 9b. It can be seen that both the π-MPPI and the MPPIwSGF are able to closely follow the terrain and encircle the target once the goal point is reached. However, if we look closely at the altitude profile in Fig. 10, it can be seen that the MPPIwSGF violates the minimum and maximum altitude



(a) Example terrain. (b) Example 3D trajectories.

Fig. 9: Example trajectories taken by MPPIwSGF and π-MPPI for the terrain following scenario.

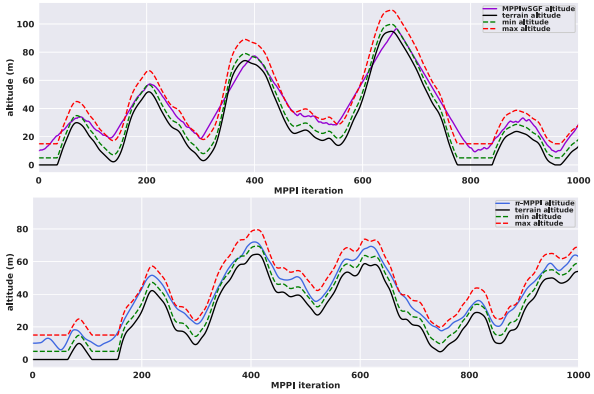
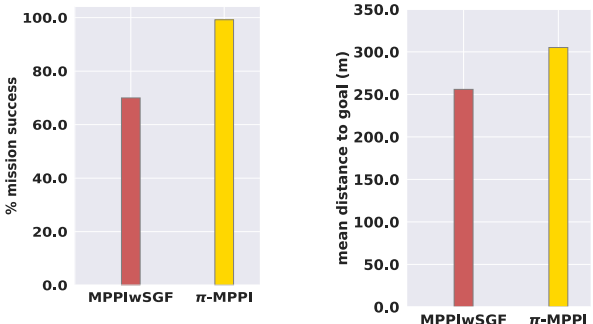


Fig. 10: Altitude profile for the MPPIwSGF and π-MPPI for the terrain following scenario.

limits in some places and also collides with the terrain sometimes. For this scenario, a successful mission is defined as the case where the FWV did not collide with the terrain along the trajectory. Fig. 11a shows the success rates for the π-MPPI and the baseline MPPIwSGF. We can see an improvement of 30% in the success rate for the π-MPPI compared to the MPPIwSGF. The mean distance from the target is similar for both the π-MPPI and the MPPIwSGF, as shown in Fig. 11b. There is no reduction in the mean distance to the goal for the π-MPPI since it's taking alternate (sometimes longer) routes to minimize the input constraint



(a) success % of not colliding with the terrain.

(b) Mean distance to goal

Fig. 11: Comparison of success rate and mean distance to the goal of MPPIwSGF and π-MPPI for the terrain following scenario.

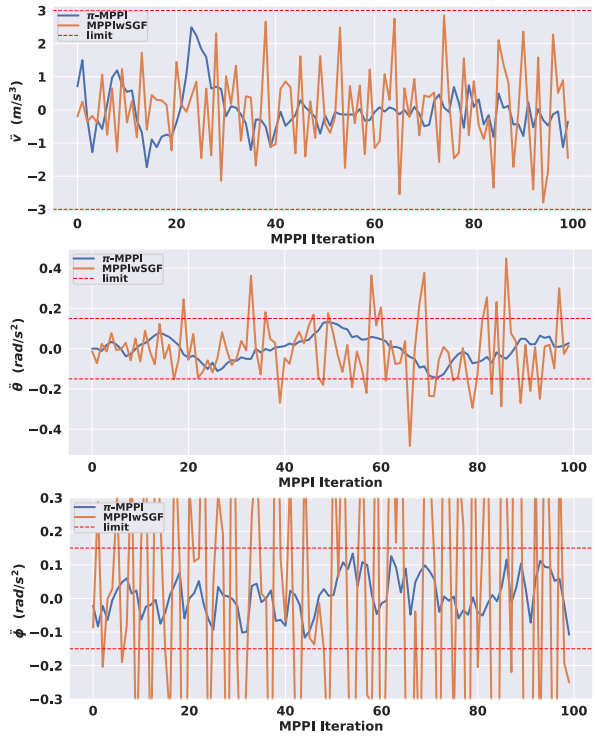


Fig. 12: Comparison of the double derivative of control inputs of MPPIwSGF and π-MPPI for an example terrain following scenario.

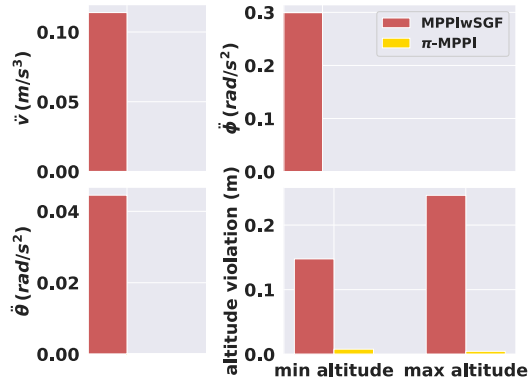


Fig. 13: Comparison of the mean constraint violations of $\ddot{v}, \ddot{\phi}, \ddot{\theta}$, min. altitude, max. altitude of MPPIwSGF and π-MPPI for the terrain following scenario.

violations. Even though the mean distance to the goal is slightly lower for the MPPIwSGF, the input constraints are violated significantly, as shown in Fig. 12. We can see that the MPPIwSGF has serious violations in $\ddot{\theta}$ and $\ddot{\phi}$. Fig. 13 shows the mean constraint violations of $\ddot{v}, \ddot{\phi}, \ddot{\theta}$, minimum altitude, and maximum allowed altitudes. The violations are almost zero for the π-MPPI whereas the MPPIwSGF shows significant violations with more than 0.10 m/s^3 for the jerk, 0.3 rad/s^2 for the $\ddot{\phi}$, and more than 0.15 m for the min. and max. altitudes.

VI. Conclusions

We presented a projection-based MPPI algorithm (π-MPPI) for finding the control inputs of fixed-wing aerial vehicles (FWV) for executing diverse tasks. The efficacy of the proposed scheme was tested using two application scenarios. Using the π-MPPI algorithm, the FWV was able to i) encircle a target in 3D space while avoiding static obstacles and ii) follow an uneven terrain closely while encircling a target. We also showed the improvements the π-MPPI algorithm brings compared to a baseline MPPIwSGF algorithm. Future work will involve the use of learning methods to accelerate the convergence of the projection QP.

References

[1] 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.

[2] I. S. Mohamed, K. Yin, and L. Liu, "Autonomous navigation of agvs in unknown cluttered environments: log-mppi control strategy," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 240–10 247, 2022.

[3] T. Kim, G. Park, K. Kwak, J. Bae, and W. Lee, "Smooth model predictive path integral control without smoothing," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 406–10 413, 2022.

[4] M. A. Olivares-Mendez, C. Fu, S. Kannan, H. Voos, and P. Campoy, "Using the cross-entropy method for control optimization: A case study of see-and-avoid on unmanned aerial vehicles," in *22nd Mediterranean conference on control and automation*. IEEE, 2014, pp. 1183–1189.

[5] K. Chepuri and T. Homem-De-Mello, "Solving the vehicle routing problem with stochastic demands using the cross-entropy method," *Annals of Operations Research*, vol. 134, pp. 153–181, 2005.

[6] M. Kobilarov, "Cross-entropy motion planning," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 855–871, 2012.

[7] J. Suh, K. Cho, and S. Oh, "Efficient graph-based informative path planning using cross entropy," in *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 5894–5899.

[8] K. Huang, S. Lale, U. Rosolia, Y. Shi, and A. Anandkumar, "Cem-gd: Cross-entropy method with gradient descent planner for model-based reinforcement learning," *arXiv preprint arXiv:2112.07746*, 2021.

[9] M. Kazim, J. Hong, M. G. Kim, and K. K. Kim, "Recent advances in path integral control for trajectory optimization: An overview in theoretical and algorithmic perspectives," *Annual Reviews in Control*, vol. 57, p. 100931, 2024.

[10] E.-T. Jeong and C.-H. Lee, "Path-following guidance using model predictive path integral control," in *Asia-Pacific International Symposium on Aerospace Technology*. Springer, 2021, pp. 313–326.

[11] E.-T. Jeong, S.-D. Lee, K.-M. Na, and C.-H. Lee, "Mppi control-based adaptive pursuit guidance for path-following control of quadrotors in the presence of wind disturbances," in *International Conference on Robot Intelligence Technology and Applications*. Springer, 2022, pp. 37–48.

[12] J. Pravitra, K. A. Ackerman, C. Cao, N. Hovakimyan, and E. A. Theodorou, "L1-adaptive mppi architecture for robust and agile control of multirotors," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 7661–7666.

[13] M. D. Houghton, A. B. Oshin, M. J. Acheson, E. A. Theodorou, and I. M. Gregory, "Path planning: Differential dynamic programming and model predictive path integral control on vtol aircraft," in *AIAA SCITECH Forum*, 2022, p. 0624.

[14] J. Pravitra, E. Theodorou, and E. N. Johnson, "Flying complex maneuvers with model predictive path integral control," in *AIAA Scitech Forum*, 2021, p. 1957.

[15] R. W. Beard and T. W. McLain, *Small unmanned aircraft: Theory and practice*. Princeton university press, 2012.

[16] 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.

[17] "Jax," https://github.com/google/jax.

[18] "Pytorch," https://pytorch.org/.