IEEE Robotics and Automation Letters (RAL) paper presented at
2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

PRIEST: Projection Guided Sampling-Based Optimization For Autonomous Navigation

Fatemeh Rastgar[1], Houman Masnavi[2], Basant Sharma[1], Alvo Aabloo[1], Jan Swevers[3], Arun Kumar Singh[1]

***Abstract*—Efficient navigation in unknown and dynamic environments is crucial for expanding the application domain of mobile robots. The core challenge stems from the non-availability of a feasible global path for guiding optimization-based local planners. As a result, existing local planners often get trapped in poor local minima. In this paper, we present a novel optimizer that can explore multiple homotopies to plan high-quality trajectories over long horizons while still being fast enough for real-time applications. We build on the gradient-free paradigm by augmenting the trajectory sampling strategy with a projection optimization that guides the samples toward a feasible region. As a result, our method can recover from the frequently encountered pathological cases wherein all the sampled trajectories lie in the high-cost region. We push the state-of-the-art (SOTA) in the following respects. Over the navigation stack of the Robot Operating System (ROS), we show an improvement of 7-13% in success rate and up to two times in total travel time metric. On the same benchmarks and metrics, our approach achieves up to 44% improvement over model predictive path integral (MPPI) and its recent variants. On simple point-to-point navigation tasks, our optimizer is up to two times more reliable than SOTA gradient-based solvers, as well as sampling-based approaches such as the Cross-Entropy Method (CEM) and VPSTO. Codes: https://github.com/fatemeh-rastgar/PRIEST**

***Index Terms*—Optimization and Optimal Control, Autonomous Vehicle Navigation, Collision Avoidance**

I. Introduction

SMOOTH and collision-free navigation in unknown and dynamic environments is challenging. The prior computed global plan invariably becomes infeasible during navigation and can no longer guide the local planner toward safe state-space regions. One possible workaround is to make the local planners themselves capable of planning over long horizons while exploring multiple homotopies in real-time. In the context of our work, homotopy refers to the different ways a robot can move around the obstacle (e.g, avoid from top or bottom in Fig.1(d)) [1]. Our work is geared towards imparting such capabilities to mobile robots.

In this paper, we consider optimization-based local planners because of their ability to satisfy constraints and produce smooth motions. There are two broad classes of approaches for trajectory optimization. On one end of the spectrum, we have gradient-based approaches [2], [3] that require the cost and constraint functions to be differentiable. Typically, these methods require a good initialization trajectory, which could be difficult to obtain in fast-changing environments.

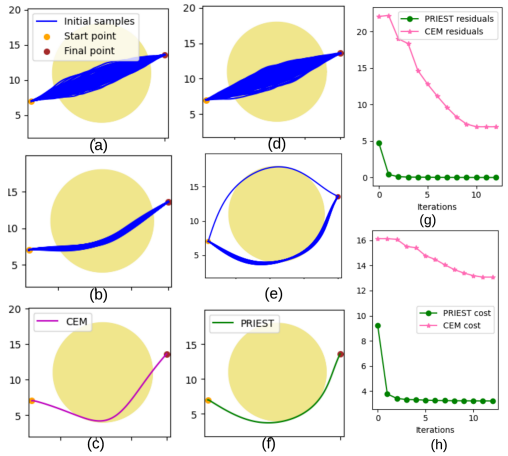

Fig. 1: A comparison of CEM (left column) and our approach PRIEST (middle column). Fig.(a)-(c) shows how a typical CEM (or any sampling-based optimizer) struggles when all the sampled initial trajectories lie in the infeasible (high-cost) region. Our approach, PRIEST, embeds a projection optimizer within any standard sampling-based approach that pushes the samples toward feasible regions before evaluating their cost. Fig.(g)-(h) presents how the cost function values and constraint residuals change across both CEM and PRIEST iterations. It is important to note that while increasing initial exploration variance might help baseline CEM in certain situations, such heuristics lack reliability across all environments, particularly in dynamic and unknown environments.

On the other end of the spectrum, we have sampling-based planners such as CEM [4] and Covariance Matrix Adaptation-Evolution Strategy (CMA-ES) [5]. These optimizers perform a random sampling of the state-space trajectories to obtain a locally optimal solution. Due to this exploration property, they can often come up with better solutions than purely gradient-based approaches [6]. However, these optimizers typically fail when all the sampled trajectories lie in the infeasible (high-cost) region (see Fig.1(a-c)).

Our main motivation is to combine the benefits of both sampling-based and gradient-based approaches. Existing efforts in this direction are mostly restricted to using sampling-based optimizers to compute a good initialization trajectory, which is subsequently fed to the gradient-based solver [7]. Our experiments in Section V-C3 show that such approaches do not work reliably in difficult benchmarks.

Our main idea is to use gradient-based approaches to improve the inner working of sampling-based optimization. We formulate a projection optimization to guide the sampling

Manuscript received September 07, 2023; Revised December 02, 2023; Accepted January 03, 2024. This letter was recommended for publication by Editor L. Pallottino upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by the European Social Fund via ICT program measure and grants PSG753 from Estonian Research Council.

[1]F. Rastgar, B. Sharma, A. Aabloo, and A. K. Singh are with the Institute of Science and Technology, University of Tartu. fatemeh@ut.ee, basant.sharma@ut.ee, alvo.aabloo@ut.ee, arun.singh@ut.ee [2]H. Manavi is with Toronto Metropolitan University. houman@torontomu.ca [3]J. Swevers is with the Mechanical Engineering department at KU Leuven University and Flanders Make KU Leuven. jan.swevers@kuleuven.be

Digital Object Identifier (DOI): see top of this page.

process toward feasible(low-cost) regions at each iteration. Thus, our approach can recover from the pathological cases where all sampled trajectories are infeasible, e.g., due to violation of collision constraints(see Fig.1(d-f)). Our core innovations and their benefits are summarized below.

Algorithmic Contribution: We present Projection Guided Sampling Based Optimization (PRIEST). The key building block is a new optimizer that can take a set of trajectories and project them onto the feasible set. We show how our projection optimizer can be effectively parallelized and accelerated over GPUs by reformulating the underlying collision and kinematic constraints. Moreover, it also naturally integrates with decentralized variants of sampling-based optimizers [8], wherein multiple sampling distributions are refined in parallel to improve the optimality of the solution. See Section IV for a summary of contributions over authors' prior works.

Improvement over the SOTA: We show that PRIEST outperforms existing approaches in terms of success rate, time-to-reach the goal, computation time, etc. In particular, we show at least 7% improvement over the ROS Navigation stack in success rate on the BARN dataset [9] while reducing the travel time by a factor of two. On the same benchmarks, our success rate is at least 35% better than SOTA local sampling-based optimizers like MPPI [10] and log-MPPI [11]. PRIEST also substantially outperforms the success rate of gradient-based solvers like ROCKIT [2](a collection of optimizers like IPOPT, ACADO, etc) and FATROP [3], and sampling-based methods CEM and VPSTO [5] in different benchmarks. Both CEM and VPSTO roll constraints as penalties into the cost functions. Thus, these baselines act as ablation experiments to further validate the role of our projection optimizer within sampling-based optimizers.

II. Problem Formulation

Symbols and Notations: Small case letters with regular and bold font represent scalars and vectors, respectively. Matrices have upper-case bold fonts. The variables t and T are time stamps and transpose. The number of planning steps, obstacles, decision variables, iterations, samples, and decentralized distributions are shown as $n_p, n_o, n_v, N, N_b, N_d$. The left superscript k denotes the trajectory optimizer's iteration. The rest of the symbols will be defined in the first place of use.

Differential Flatness: We assume $\mathbf{u} = \mathbf{\Phi}(x^{(q)}(t), y^{(q)}(t), z^{(q)}(t))$: the control inputs can be obtained post-hoc through some analytical mapping $\mathbf{\Phi}$ of q^{th} level derivatives of the position-level trajectory. For example, the forward velocity and curvature control input for a car-like robot can be expressed as $\mathbf{\Phi}=(\sqrt{\dot{x}(t)^2+\dot{y}(t)^2}, \frac{\ddot{y}(t)\dot{x}(t)-\ddot{x}(t)\dot{y}(t)}{(\dot{x}(t)^2+\dot{x}(t)^2)^{1.5}})$. The motivation for leveraging differential flatness is to eliminate the explicit need of having non-linear equality constraints stemming from motion model (e.g see [12]). This, in turn will be crucial for inducing parallelized and GPU accelerated structure in PRIEST.

Trajectory Optimization: We are interested in solving the following 3D trajectory optimization:

$$\min_{x(t),y(t),z(t)} \; c_1(x^{(q)}(t), y^{(q)}(t), z^{(q)}(t)), \quad \text{(1a)}$$

$$x^{(q)}(t), y^{(q)}(t), z^{(q)}(t)|_{t=t_0} = \mathbf{b}_0, \; x^{(q)}(t), y^{(q)}(t), z^{(q)}(t)|_{t=t_f} = \mathbf{b}_f, \quad \text{(1b)}$$

$$\dot{x}^2(t)+\dot{y}^2(t)+\dot{z}^2(t) \leq v_{max}^2, \; \ddot{x}^2(t)+\ddot{y}^2(t)+\ddot{z}^2(t) \leq a_{max}^2, \quad \text{(1c)}$$

$$s_{min} \leq (x(t), y(t), z(t)) \leq s_{max} \quad \text{(1d)}$$

$$-\frac{(x(t)-x_{o,j}(t))^2}{a^2} - \frac{(y(t)-y_{o,j}(t))^2}{a^2} - \frac{(z(t)-z_{o,j}(t))^2}{b^2} + 1 \leq 0, \quad \text{(1e)}$$

where $(x(t), y(t), z(t))$ and $(x_{o,j}(t), y_{o,j}(t), z_{o,j}(t))$ respectively denote the robot and the j^{th} obstacle position at time t. The function $c_1(.)$ is defined in terms of derivatives of the position-level trajectories and can encompass commonly used penalties on accelerations, velocities, curvature, etc. We can also leverage differential flatness to augment control costs in $c_1(.)$ as well. The affine inequalities (1d) model bounds on the robot workspace. The vectors $\mathbf{b}_0$ and $\mathbf{b}_f$ in (1b) represent the initial and final values of boundary condition on the q^{th} derivative of the position-level trajectory. In our formulation, $q=\{0, 1, 2\}$. Inequalities (1c) denotes the velocity and acceleration bounds with their respective maximum values being v_{max} and a_{max}. In (1e), we enforce collision avoidance, assuming obstacles are modeled as axis-aligned ellipsoids with dimensions (a, a, b).

Remark 1. *The cost functions $c_1(.)$ need not be convex, smooth or even have an analytical form in our approach.*

Trajectory Parametrization and Finite Dimensional Representation : To ensure smoothness in the trajectories, we parametrize the optimization variables $(x(t), y(t), z(t))$ as

$$\begin{bmatrix} x(t_1) \\ \vdots \\ x(t_{n_p}) \end{bmatrix}^T = \mathbf{P}\ \mathbf{c}_x, \begin{bmatrix} y(t_1) \\ \vdots \\ y(t_{n_p}) \end{bmatrix}^T = \mathbf{P}\mathbf{c}_y, \begin{bmatrix} z(t_1) \\ \vdots \\ z(t_{n_p}) \end{bmatrix}^T = \mathbf{P}\ \mathbf{c}_z \quad (2)$$

where $\mathbf{P}$ is a matrix created using polynomial basis functions that are dependent on time and $\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z$ represent the coefficients of the polynomial. The expression remains applicable for derivatives by utilizing $\dot{\mathbf{P}}$ and $\ddot{\mathbf{P}}$.

By incorporating the parametrized optimization variables stated in (2) and compact representation of variables, we can reframe the optimization problem (1a)-(1e) as follows:

$$\min_{\boldsymbol{\xi}} c_1(\boldsymbol{\xi}) \quad \text{(3a)}$$

$$\mathbf{A}\boldsymbol{\xi} = \mathbf{b}_{eq} \quad \text{(3b)}$$

$$\mathbf{g}(\boldsymbol{\xi}) \leq \mathbf{0}, \quad \text{(3c)}$$

where $\boldsymbol{\xi}=\begin{bmatrix}\mathbf{c}_x^T\ \mathbf{c}_y^T\ \mathbf{c}_z^T\end{bmatrix}^T$. With a slight abuse of notation, we have now used $c_1(.)$ to denote a cost function dependent on $\boldsymbol{\xi}$. The matrix $\mathbf{A}$ is a block diagonal where each block on the main diagonal consists of $\begin{bmatrix}\mathbf{P}_0\ \dot{\mathbf{P}}_0\ \ddot{\mathbf{P}}_0\ \mathbf{P}_{-1}\end{bmatrix}$. The subscript $0, -1$ signify the first and last row of the respective matrices and pertain to the initial and final boundary constraints. The vector $\mathbf{b}_{eq}$ is simply the stack of $\mathbf{b}_0$ and $\mathbf{b}_f$. The function $\mathbf{g}$ contains all the inequality constraints (1c)-(1e).

III. Main Algorithmic Results

This section presents our main algorithmic contributions. An overview of our approach is shown in Fig.2. The main differentiating factor from existing baselines lies in the insertion of the projection optimizer between the sampling and cost evaluation block.

We next present our main building block: the projection optimizer, followed by its integration into a sampling-based optimizer.

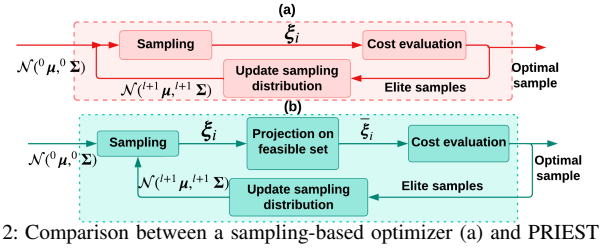

Fig. 2: Comparison between a sampling-based optimizer (a) and PRIEST (b)

A. Projection Optimization

Consider the following optimization problem

$$\min_{\bar{\boldsymbol{\xi}}_i} \frac{1}{2}\|\bar{\boldsymbol{\xi}}_i - \boldsymbol{\xi}_i\|_2^2, \ i = 1, 2, ..., N_b \tag{4}$$

$$\mathbf{A}\bar{\boldsymbol{\xi}}_i = \mathbf{b}_{eq}, \qquad \mathbf{g}(\bar{\boldsymbol{\xi}}_i) \leq \mathbf{0} \tag{5}$$

The cost function (4) aims to minimally modify the i^{th} sampled trajectory $\boldsymbol{\xi}_i$ to $\bar{\boldsymbol{\xi}}_i$ in order to satisfy the equality and inequality constraints. In Section VII, we show that for a certain class of constraint functions $\mathbf{g}$ formed with quadratic and affine constraints, optimization (4)-(5) can be reduced to the fixed-point iteration of the following form.

$$^{k+1}\mathbf{e}_i, \, ^{k+1}\boldsymbol{\lambda}_i = \mathbf{h}(^{k}\bar{\boldsymbol{\xi}}_i, \, ^{k}\boldsymbol{\lambda}_i) \tag{6a}$$

$$^{k+1}\bar{\boldsymbol{\xi}}_i = \arg\min_{\bar{\boldsymbol{\xi}}_i} \frac{1}{2}\|\bar{\boldsymbol{\xi}}_i - \boldsymbol{\xi}_i\|_2^2 + \frac{\rho}{2}\left\|\mathbf{F}\bar{\boldsymbol{\xi}}_i - \, ^{k+1}\mathbf{e}_i\right\|_2^2 - \, ^{k+1}\boldsymbol{\lambda}_i^T\bar{\boldsymbol{\xi}}_i,$$

$$\mathbf{A}\bar{\boldsymbol{\xi}}_i = \mathbf{b}_{eq} \tag{6b}$$

In (6a)-(6b), $\mathbf{F}$ represents a constant matrix and $\mathbf{h}$ is some closed-form analytical function. The vector $^{k+1}\boldsymbol{\lambda}_i$ is the Lagrange multiplier at iteration $k+1$ of the projection optimization, and the vector $\mathbf{e}_i$ is defined in details in Section VII. We derive these entities in Section VII. The main computational burden of projection optimization stems from solving the QP (6a). However, since there are no inequality constraints in (6b), the QP essentially boils down to an affine transformation of the following form:

$$(^{k+1}\bar{\boldsymbol{\xi}}_i, \, ^{k+1}\boldsymbol{\nu}_i) = \mathbf{M}\boldsymbol{\eta}(^{k}\bar{\boldsymbol{\xi}}_i), \tag{7a}$$

$$\mathbf{M} = \begin{bmatrix} \mathbf{I} + \rho\mathbf{F}^T\mathbf{F} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix}^{-1}, \boldsymbol{\eta} = \begin{bmatrix} -\rho\mathbf{F}^T \, ^{k+1}\mathbf{e}_i + \, ^{k+1}\boldsymbol{\lambda}_i + \boldsymbol{\xi}_i \\ \mathbf{b}_{eq} \end{bmatrix} \tag{7b}$$

where $\boldsymbol{\nu}_i$ represents the dual variables associated with the equality constraints [13], [14].

GPU Accelerated Batch Operation: Our approach requires projecting several sampled trajectories onto the feasible set (see Fig.2). However, this can be computationally prohibitive if done sequentially. Fortunately, our projection optimizer has certain structures that allow for batch/parallelized operation. To understand this further, note that the matrix $\mathbf{M}$ in (7a) is independent of the input trajectory sample $\boldsymbol{\xi}_i$. In fact, $\mathbf{M}$ remains the same, irrespective of the trajectory sample that we need to project to the feasible set. This allows us to express the solution (7a), $\forall \boldsymbol{\xi}_i, i = 1, 2, \ldots, N_b$ as one large matrix-vector product, that can be trivially parallelized over GPUs. Similarly, acceleration can be done for the function $\mathbf{h}$ that consists of just element-wise products and sums.

Scalability: The matrix $\mathbf{M}$ in (7a) needs to be computed only once as $\mathbf{A}, \mathbf{F}$ do not change across the projection iteration. The matrix $\boldsymbol{\eta}$ on the other hand, is recomputed at every iteration and its computation cost is primarily dominated by $\mathbf{F}^T \, ^{k+1}\mathbf{e}_i$. The number of rows in $\mathbf{F}$ and $\mathbf{e}$ increases linearly with planning horizon, number of obstacles or batch size (see Remark 5). This feature, coupled with GPU acceleration,

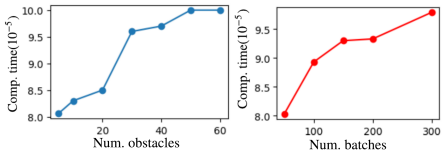

Fig. 3: Scalability of per-iteration computation time of our projection optimizer with respect to number of obstacles and batch size

provides excellent scalability to our approach for long-horizon planning in highly cluttered environments. Fig.3 shows how the average per-iteration time of the projection optimizer scales with the number of obstacles and batch size. Both graphs reflect the typical scaling pattern of matrix multiplication on GPUs [15].

B. Projection Guided Sampling-Based Optimizer

1) Algorithm Description: Algorithm 1 presents the other core contribution of this paper. It starts by generating N_b samples of polynomial coefficients $\boldsymbol{\xi}$ from a Gaussian distribution with $\mathcal{N}(^{l}\boldsymbol{\mu}, ^{l}\boldsymbol{\Sigma})$ at iteration $l = 0$ (line 3). The sampled $\boldsymbol{\xi}_i$ is then projected onto the feasible set (lines 4-5). Due to real-time constraints, it may not be possible to run the projection optimization long enough to push all the sampled trajectories to feasibility. Thus, in line 6, we compute the constraint residuals $r(\bar{\boldsymbol{\xi}}_i)$ associated with each sample. In line 7, we select the top N_{proj} samples with the least constraint residuals and append them to the list $ConstraintEliteSet$. In lines 8-9, we construct an appended cost, c_{aug}, by appending the residual to the primary cost function. We evaluate c_{aug} on the $ConstraintEliteSet$ samples. In line 11, we once again select the top N_{elite} samples with the lowest c_{aug} and append them to the list $EliteSet$. Finally, in line 12, we update the distribution based on the samples of the $EliteSet$ and the associated c_{aug} values. The final output of the optimizer is the sample from the $EliteSet$ with the lowest c_{aug}.

2) Updating the Sampling Distribution: There are several ways to perform the distribution update in line 13 of Alg. 1. We adopt the following MPPI-like updated rule [16].

$$^{l+1}\boldsymbol{\mu} = (1-\sigma) \, ^{l}\boldsymbol{\mu} + \sigma\left(\frac{1}{\sum_{m\in C} c_m}\right) \sum_{m\in C} \bar{\boldsymbol{\xi}}_m c_m, \tag{9a}$$

$$^{l+1}\boldsymbol{\Sigma} = (1-\sigma) \, ^{l}\boldsymbol{\Sigma} + \sigma \frac{\sum_{m\in C} c_m (\bar{\boldsymbol{\xi}}_m - \, ^{l+1}\boldsymbol{\mu})(\bar{\boldsymbol{\xi}}_m - \, ^{l+1}\boldsymbol{\mu})^T}{\sum_{m\in C} c_m}, \tag{9b}$$

$$c_m = \exp\left(\gamma^{-1}(c_{aug}(\bar{\boldsymbol{\xi}}_m) - \delta)\right), \tag{9c}$$

where the scalar constant σ is the so-called learning rate. The set C consists of the top N_{elite} selected trajectories (line 11). The constant γ specifies the sensitivity of the exponentiated cost function $c_{aug}(\bar{\boldsymbol{\xi}}_m)$ for top selected trajectories. $\delta = \min c_{aug}(^{l}\bar{\boldsymbol{\xi}}_m)$ is defined to prevent numerical instability.

Remark 2. *PRIEST employs dual sorting (line 7, 11 in Alg.1) to account for the fact that under real-time constraints, we are likely to run the projection optimizer only for a few iterations. Thus, in that limited time not all trajectory samples would be close to the feasible region. A similar dual sorting approach is also presented in [17], although without the use of any projection optimizer. The baseline CEM in Section V essentially follows [17] and we show that without a dedicated*

Algorithm 1: Projection Guided Sampling-Based Optimization (PRIEST)

Input: Initial states
Initialization: Initiate ${}^{l}\boldsymbol{\mu}$ and ${}^{l}\boldsymbol{\Sigma}$ at $i = 0$
1 **for** $l \leq N$ **do**
2 Initialize $CostList = []$
3 Draw N_b samples $\boldsymbol{\xi}_1, ..., \boldsymbol{\xi}_{N_b}$ from $\mathcal{N}({}^{l}\boldsymbol{\mu}, {}^{l}\boldsymbol{\Sigma})$ // Generating samples from a Gaussian distribution
4 5 Solve the inner convex optimizer to obtain $\bar{\boldsymbol{\xi}}_i$ // Pushing the samples into feasible trajectory space

$$\min_{\bar{\boldsymbol{\xi}}_i} \frac{1}{2} \left\| \bar{\boldsymbol{\xi}}_i - \boldsymbol{\xi}_i \right\|_2^2, \tag{8a}$$

$$\mathbf{A}\bar{\boldsymbol{\xi}}_i = \mathbf{b}_{eq} \tag{8b}$$

$$\mathbf{g}(\bar{\boldsymbol{\xi}}_i) \leq \mathbf{0}, \tag{8c}$$

6 Compute the residuals set $r(\bar{\boldsymbol{\xi}}_i)$ //getting residuals to know how well the constraints are satisfied
7 $ConstraintEliteSet \leftarrow$ Select N_{proj} samples from $r(\bar{\boldsymbol{\xi}}_i)$ with the lowest values. // Select top samples with the lowest residuals
8 Evaluate the cost
9 $c_{aug} \leftarrow c_1(\bar{\boldsymbol{\xi}}_i) + r(\bar{\boldsymbol{\xi}}_i)$ // Cost evaluation
10 Append $cost$ to the $CostList$
11 $EliteSet \leftarrow$ Select N_{elite} top samples with the lowest cost obtained from the $CostList$. //Select top samples which influence sampling distribution
12 Update the new mean and covariance, ${}^{l+1}\boldsymbol{\mu}$ and ${}^{l+1}\boldsymbol{\Sigma}$, using (9a)-(9b) // Update mean and covariance based on the top samples
13 **end**
14 **Return** $\bar{\boldsymbol{\xi}}_i$ corresponding to the lowest cost in $EliteSet$

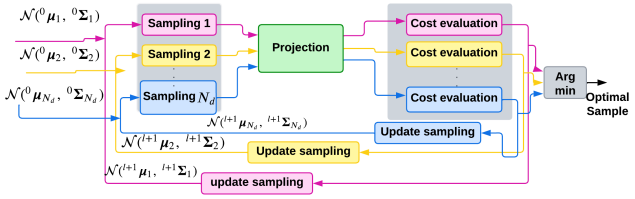

Fig. 4: Decenteralized variant of PRIEST inspired by [8], wherein we instantiate and maintain different N_d Gaussian distributions in parallel to counter poor local minima. An important thing to note is that our projection optimizer naturally fits into the decentralized structure.

projection optimizer, the dual-sorting by itself is not sufficient for reliable performance.

Remark 3. *Alg.1 is agnostic to the distribution update rule; e.g,* (9b) *can be replaced with CMA-ES style update.*

C. Decenteralized PRIEST (D-PRIEST)

Our main objective in this section is to show that our projection optimizer naturally integrates into decentralized optimizers built along the lines of [8] that allows for sampling from a more diverse distribution.

Fig.4 shows our proposed approach. We initialize N_d different Gaussian distributions ${}^{l}\boldsymbol{\mu}_j, {}^{l}\boldsymbol{\Sigma}_j$ at $l = 0$. We sample $\frac{N_b}{N_d}$ samples of $\boldsymbol{\xi}$ from each of these distributions. The sampled $\boldsymbol{\xi}_{ij}$ (i^{th} sample from j^{th} Gaussian) are then stacked row-wise to form a matrix. Importantly, such a construction allows us to easily track which row of the matrix corresponds to samples from which of the N_d Gaussian distributions. The projection optimizer then simultaneously guides all the samples toward feasible regions. The output from the projection is then separated back into N_d sets on which the cost functions are evaluated in parallel. We then update the sampling distribution in parallel based on the cost values. Finally, after l iterations, the optimal trajectory from each optimizer instance is compared and the one with the lowest cost is selected as the optimal solution.

IV. Connections to Existing Works

Connection to CEM-GD: Alternate approaches of combining sampling and gradient-based approach were presented recently in [4], [18]. In these two cited works, the projection at line 5 of Alg.1 is replaced with a gradient step of the form $\bar{\boldsymbol{\xi}}_i = \boldsymbol{\xi}_i - \sigma \nabla_{\boldsymbol{\xi}} c_1$, for some learning-rate σ. Our approach PRIEST improves [4], [18] in two main aspects. First, it can be applied to problems with non-smooth and non-analytical cost functions. Second, the gradient-descent-based can be computationally slow as it relies on taking small steps toward optimal solutions. In contrast, the projection optimizer in Alg.1 leverages convex optimization, specifically quadratic programming to ensure faster convergence.

Exploration Strategy: Sampling-based optimizers like MPPI [10] and its variants [11], [19] explore by injecting random perturbation into the control inputs to obtain a trajectory distribution. PRIEST, on the other hand, injects perturbation in the parameter space (polynomial coefficients), leading to one core advantage. We can inject larger perturbations into the parameter space that helps in better exploration over longer horizons (see Fig.5(a)-(b)). Moreover, the projection optimizer ensures that the trajectory distribution satisfies boundary constraints and is pushed toward the feasible region. In contrast, increasing the covariance of control perturbation has been shown to make MPPI diverge [11]. Adapting the co-variance matrix within MPPI can also lead to potential benefits in exploration. However, existing works such as [20] in this regard haven't yet been tested for navigation in unknown, cluttered and dynamic environments.

Improvement over Author's Prior Work [21]: PRIEST builds on our prior work [21] that used projection-augmented sampling for visibility-aware navigation. Our current work targets a much broader scope of navigation problems with potentially non-smooth costs. On the algorithmic side, we extended the projection optimizer of [21] to 3D (see Section VII) and improved the distribution update rule to account for actual cost values. Moreover, the decentralized variant of PRIEST is also a major contribution over [21]. On the experimental side, we present a more elaborate benchmarking with several existing methods on both open-source as well as custom data sets.

V. Validation and Benchmarking

A. Implementation Details

We developed Alg.1, PRIEST, in Python using JAX [22] library as our GPU-accelerated algebra backend. The simulation framework for our experiments was built on top of the ROS

[23] and used the Gazebo physics simulator. All benchmarks were executed on a Legion7 Lenovo laptop equipped with an Intel Core i7 processor and an Nvidia RTX 2070 GPU. We used the open3d library to downsampled PointCloud data [24]. We chose $t_f = 10, N = 13, N_b = 110,\ N_{proj} = 80$ and N_{elite}=20. All the compared baselines operated with the same planning horizon. Moreover, to give the best possible chance MPPI, log-MPPI, they were run with a sample size of 2496, which is more than 20 times higher than that used by PRIEST. The temperature parameter for MPPI, log-MPPI was 0.572, while PRIEST used 0.9 for the same parameter. The sample size of CEM and VPSTO was same as PRIEST. Obstacles are detected using LIDAR in both real-world experiments and simulations.

1) Baselines and Benchmarks: We compared our approach with different baselines in three sets of benchmarks.

Comparison on BARN Dataset [9]: This benchmark requires a mobile robot to iteratively plan to navigate an obstacle field in a receding horizon fashion. We used the BARN dataset that contains 300 environments with varied levels of complexity, specifically designed to create local-minima traps for the robot. We evaluate our approach against DWA [25], TEB [1] implemented in the ROS navigation stack, MPPI [10], and log-MPPI [11]. All baselines along with PRIEST had access to only the local cost map or point cloud. TEB and DWA used a combination of graph-search and optimization while MPPI and log-MPPI are purely optimization-based approaches. We used a holonomic robot modeled as a double-integrator system in this benchmark. Consequently, the differential flatness function to extract control inputs was $\mathbf{\Phi}$=$(\ddot{x}(t), \ddot{y}(t))$. The cost function ($c_1$) had the following form:

$$\sum_t \ddot{x}(t)^2 + \ddot{y}(t)^2 + c_\kappa + c_p \tag{10}$$

where c_κ=$(\frac{\dddot{y}(t)\dot{x}(t) - \ddot{x}(t)\dddot{y}(t)}{(\dot{x}(t)^2 + \dot{x}(t)^2)^{1.5}})^2$ penalizes curvature and c_p minimizes the orthogonal distance of the computed trajectory from a desired straight-line path to the goal. Note that c_p does not have an analytical form as it requires computing the projection of a sampled trajectory way-point onto the desired path.

Point to Point Navigation with Differentiable Cost: In this benchmark, we considered the task of generating a single trajectory between a start and a goal location. The cost function c_1 consisted of first term from (10). For comparison, we considered SOTA gradient-based optimizers ROCKIT [2] and FATROP [3] and sampling-based optimizers CEM, and VPSTO [5]. We designed 50 cluttered environments wherein obstacles are placed randomly in 2D and 3D spaces (see Fig.6). Around 100 trials were performed in both 2D and 3D.

Comparison in a Dynamic Environment: We compare PRIEST with CEM, log-MPPI, MPPI, TEB, and DWA in dynamic environments, utilizing the same cost function as used for the BARN dataset (10). In this benchmark, we introduced ten obstacles, each with a velocity of $0.1m/s$, moving in the opposite direction of the robot. We run simulations across 30 different obstacle configurations and velocities.

2) Metrics: We utilize the following metrics for benchmarking against other baselines:

Success Rate: A run is considered successful when the robot approaches the final point within a 0.5m radius without any collision. The success rate is calculated as the ratio of the total number of successful runs to the overall number of runs.

Travel Time: This refers to the duration it takes for the robot to reach the vicinity of the goal point.

Computation Time: This metric quantifies the time required to calculate a solution trajectory.

B. Qualitative Results

1) A Simple Benchmark: In Fig.1, our objective is to contrast the behavior of CEM(a-c) and PRIEST(d-e) in a scenario wherein all the initial sampled trajectories lie within a high-cost/infeasible region. As can be seen, our projection optimizer effectively pushes the sample trajectories out of the infeasible region. In contrast, the CEM samples persistently remain within the infeasible region. Fig.1(g-h) provides an empirical validation of Alg.1 by showing that both constraint residuals and cost values gradually reduce and saturate as the iterations progress. The projection optimizer is essentially a set of analytical transformations over the sampled trajectories. Thus, Alg.1 retains the convergence properties of the base sampling-based optimizer upon which the projection part is embedded. For example, it will inherit the properties of CEM when integrated with CEM-like sampler. Moreover, projection optimizer by construction satisfies the boundary constraints on the trajectories and ensures reduction in the cost by pushing the samples towards feasible region.

2) Receding Horizon Planning on Barn Dataset: In Fig. 5, we show a qualitative comparison between TEB and our approach PRIEST within one of the BARN environments. Both methods can search over multiple homotopies but differ in their process. While TEB uses graph search, PRIEST relies on the stochasticity of the sampling process guided through the projection optimizer. As a result, the latter can search over a longer horizon and wider state space. It is worth pointing out that increasing the planning horizon of TEB dramatically increases the computation time and thus degrades the overall navigation performance instead of improving it. We present the quantitative comparison with TEB and other baselines in the next subsection.

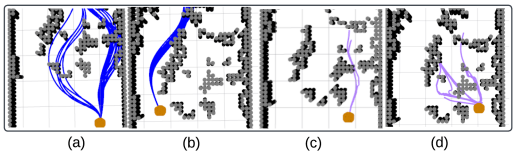

Fig. 5: Qualitative result of MPC built on PRIEST (a-b) and TEB (c-d). Blue and purple trajectories show top samples in the PRIEST and TEB planner.

3) Point-to-Point Navigation Benchmark: Fig.6 shows trajectories generated by PRIEST alongside those generated by gradient-based optimizers, ROCKIT, FATROP, and sampling-based optimizers CEM, VPSTO. For the particular 2D example shown, both PRIEST and VPSTO successfully generated collision-free trajectories, while the other baselines failed. For the shown 3D environment, only PRIEST and CEM achieved collision-free trajectories.

4) Decenteralized Variant: Fig.7 shows the application of D-PRIEST for the trajectory planning of a car-like vehicle. The cost function c_1 penalized the magnitude of axis-wise

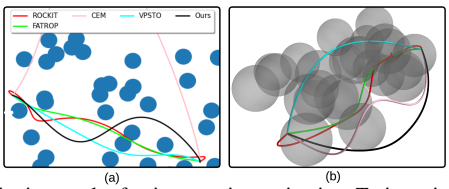

Fig. 6: Qualitative result of point-to-point navigation. Trajectories for different approaches are shown in the same color for both 2D and 3D.

accelerations and steering angle, respectively. We leveraged the differential flatness property to express steering angle as a function of axis-wise velocity and acceleration terms [12]. As can be seen from Fig.7, D-PRIEST maintained three different distributions (shown in green, red, and blue) in parallel leading to multi-modal behaviors. Intuitively, the obtained maneuvers correspond to overtaking the static obstacles (shown in black) from left to right or slowing down and shifting to another lane. In contrast, the traditional CEM was able to obtain only a single maneuver for the vehicle.

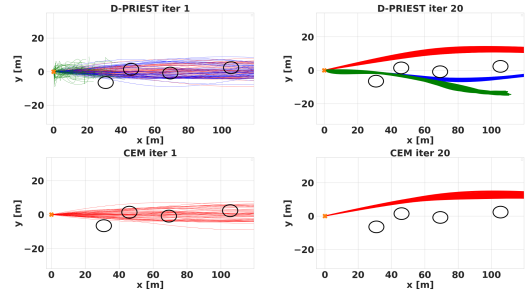

Fig. 7: Comparison of D-PRIEST with baseline CEM. As is shown, the former updates multiple parallel distributions, resulting in multi-modal optimal trajectory distribution upon convergence. D-PRIEST maintained three different distributions (shown in green, red, and blue) and thus could obtain multi-modal behavior. In contrast, CEM which only has a single distribution, provides a limited set of options for collision-free trajectories.

C. Quantiative Results

1) Comparison with MPPI, Log-MPPI, TEB, DWA: Table I summarizes the quantitative results. PRIEST reaches a 90% success rate while the best baseline (TEB) achieves 83%. The latter (and DWA as well) uses graph search along with complex recovery maneuvers for an improved success rate, albeit at the expense of increased travel time. In contrast, purely optimization-based approaches MPPI and log-MPPI perform 32% and 35% worse than PRIEST in success rate, accompanied by marginally higher travel times as well. PRIEST shows a slightly higher mean computation time but remains fast enough for real-time applications.

TABLE I: Comparisons on the BARN Dataset

Method	Success rate	Travel time (s) Mean/ Min/Max	Computation time (s) Mean/ Min/Max
DWA	76%	52.07/ 33.08/145.79	0.037/ 0.035/0.04
TEB	83%	52.34/ 42.25/106.32	0.039/0.035/0.04
MPPI	58%	36.66/ 31.15/99.62	0.019/0.018/0.02
log-MPPI	55%	36.27/30.36/58.84	0.019/0.018/0.02
PRIEST	**90%**	**33.59**/ 30.03/70.98	0.071/0.06/0.076

Remark 4. *The total samples used by PRIEST across all iteration is* $110 \times 13 = 1430$ *which is still less than that used for baseline MPPIs. Moreover, we didn't find any improvement by increasing the sample-size of MPPI and Log-MPPI.*

TABLE II: Comparing PRIEST with Gradient/Sampling-Based Optimizers

Method	Success rate	Computation time (s)(Mean/Min/Max)
ROCKIT-2D	46%	2.57/0.6/6.2
FATROP-2D	64%	0.63/0.07/2.87
PRIEST-2D	**95%**	**0.043/0.038/0.064**
CEM-2D	78%	0.017/0.01/0.03
VPSTO-2D	66%	1.63/0.78/4.5
ROCKIT-3D	65%	1.65/0.68/5
FATROP-3D	81%	0.088/0.034/0.23
PRIEST-3D	**90%**	**0.053/0.044/0.063**
CEM-3D	74%	0.028/0.026/0.033
VPSTO-3D	37%	3.5/0.93/3.5

2) Comparison with Additional Gradient-Based and Sampling-based Optimizers: Table II compares the performance of the PRIEST with all the baselines in 2D and 3D cluttered environments (recall Fig.6). ROCKIT and FATROP were initialized with simple straight-line trajectories between the start and the goal that were typically not collision-free. Due to conflicting gradients from the neighboring obstacle, both these methods often failed to obtain a collision-free trajectory. Interestingly, the sampling-based approaches didn't fare particularly better as both CEM and VPSTO reported a large number of failures. We attribute the failures of VPSTO and CEM to two reasons. First, most of the sampled trajectories for both CEM and VPSTO fell into the high-cost/infeasible area, and as discussed before, this creates a pathologically difficult case for sampling-based optimizers. Second, both CEM and VPSTO roll constraints into the cost as penalties and can be really sensitive to tuning the individual cost terms. In summary, Table II shows the importance of PRIEST that uses convex optimization to guide trajectory samples towards constraint satisfaction.

PRIEST also shows superior computation time than ROCKIT and FATROP. The CEM run times are comparable to PRIEST. Although VPSTO numbers are high, we note that the original author implementation that we use may not have been optimized for computation speed.

3) Combination of Gradient-Based and Sampling-based Optimizers: A simpler alternative to PRIEST can be just to use a sampling-based optimizer to compute a good guess for the gradient-based solvers [7]. However, such an approach will only be suitable for problems with differentiable costs. Nevertheless, we evaluate this alternative for the point-to-point benchmark of Fig.6. We used CEM to compute an initial guess for ROCKIT and FATROP. The results are summarized in Table III. As can be seen, while the performance of both ROCKIT and FATROP improved in 2D environments, the success rate of the latter decreased substantially in the 3D variant. The main reason for this conflicting trend is that the CEM (or any initial guess generation) is unaware of the exact capabilities of the downstream gradient-based optimizer. This unreliability forms the core motivation behind PRIEST, which outperforms all ablations in Table III. By embedding the projection optimizer within the sampling process itself (refer Alg.1) and augmenting the projection residual to the cost function, we ensure that the sampling and projection complement each other.

4) Benchmarking in Dynamic Environments: Table IV presents the results obtained from the experiments in dynamic environments. Herein, the projection optimizer of PRIEST

TABLE III: Comparing PRIEST with Hybrid Gradient-Sampling Baselines.

Method	Success rate	Computation time (s) (Mean/Min/Max)
ROCKIT-CEM	94%	2.07/0.69/6.0
FATROP-CEM	84%	0.34/0.06/0.96
PRIEST-2D	**95%**	**0.043/0.038/0.064**
ROCKIT-CEM	**100%**	1.19/0.7/2.56
FATROP-CEM	25%	0.056/0.039/0.079
PRIEST-3D	90%	**0.053/0.044/0.063**

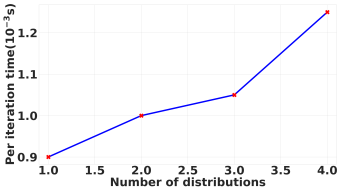

Fig. 8: Linear computation-time scaling of decentralized PRIEST with respect to the number of parallel distributions maintained at each iteration

Method	Succ. rate	Travel time(s) mean min/max
log-MPPI	60%	18.80 15.68/24.3
MPPI	53%	19.38 16.28/27.08
CEM	46%	11.95 9.57/14.21
DWA	66%	33.4 31.4/37.17
PRIEST	**83%**	**11.95 11.43/13.39**

TABLE IV: Comparisons in cluttered and dynamic environments

ensures collision constraint satisfaction with resepct to the linear prediction of the obstacles' motions. By having a success rate of 83%, our method outperforms other approaches. Furthermore, our method shows competitive efficiency with a mean travel time of 11.95 seconds. Overall, the results show the superiority of our approach in dealing with the complexities of cluttered dynamic environments, making it a promising solution for real-world applications in human-habitable environments.

5) Scaling of D-PRIEST: Fig. 8 shows the linear scaling of the per-iteration time of D-PRIEST with respect to the number of distributions. Typically, solutions are obtained in around 20 iterations. Thus, around $N_d = 4$ parallel distributions can be maintained under real-time constraints.

D. Real-world Demonstration

The accompanying video compares PRIEST with MPPI, log-MPPI, TEB, and DWA in real-world experiments. We used Clearpath Jackal equipped with Velodyne LIDAR and its state was obtained using OptiTrack motion capture system. The results are shown in website https://sites.google.com/view/priest-optimization and the accompanying video. The videos show PRIEST outperforming the considered baselines in success rate metric.

VI. Conclusions and Future Work

We presented PRIEST, an important contribution towards leveraging the benefits of both sampling-based optimizer and convex optimization. In particular, we used the latter to derive a GPU-accelerated projection optimizer that guides the trajectory sampling process. We also showed how the same projection set-up can be easily embedded within decentralized variants of sampling optimizers wherein multiple parallel distributions are maintained at each iteration. We performed extensive benchmarking showcasing the benefits of PRIEST over SOTA approaches in the context of autonomous navigation in unknown environments. One limitation of our work is that PRIEST, in its current form, is not designed to excel in complex maze-like environments. Thus, motivated by recent works like [26], we intend to imbibe some environment prediction capability within PRIEST in future works. Another avenue for improvement involves including arbitrary dynamics and control constraints within the formulation.

VII. Appendix

Reformulating constraints: We reformulate collision avoidance inequality constraints (1e), $\mathbf{f}_{o,j} = 0$, as follows:

$$\mathbf{f}_{o,j}=\begin{Bmatrix} x(t)-x_{o,j}(t)-ad_{o,j}(t)\cos\alpha_{o,j}(t)\sin\beta_{o,j}(t) \\ y(t)-y_{o,j}(t)-ad_{o,j}(t)\sin\alpha_{o,j}(t)\sin\beta_{o,j}(t) \\ z(t)-z_{o,j}(t)-bd_{o,j}(t)\cos\beta_{o,j}(t) \end{Bmatrix}, d_{o,j}(t)\geq 1 \tag{11}$$

where $d_{o,j}(t), \alpha_{o,j}(t)$ and $\beta_{o,j}(t)$ are the line-of-sight distance and angles between the robot center and j^{th} obstacle (see [13], [27], [28]).

Similarly, the inequality constraints (1c), which show the maximum velocity and acceleration, can be reformulated as:

$$\mathbf{f}_v=\begin{Bmatrix} \dot{x}(t)-d_v(t)v_{max}\cos\alpha_v(t)\sin\beta_v(t) \\ \dot{y}(t)-d_v(t)v_{max}\sin\alpha_v(t)\sin\beta_v(t) \\ \dot{z}(t)-d_v(t)v_{max}\cos\beta_v(t) \end{Bmatrix}, 0 \leq d_v(t) \leq 1, \tag{12a}$$

$$\mathbf{f}_a=\begin{Bmatrix} \ddot{x}(t)-d_a(t)a_{max}\cos\alpha_a(t)\sin\beta_a(t) \\ \ddot{y}(t)-d_a(t)a_{max}\sin\alpha_a(t)\sin\beta_a(t) \\ \ddot{z}(t)-d_a(t)a_{max}\cos\beta_a(t) \end{Bmatrix}, 0 \leq d_a(t) \leq 1, \tag{12b}$$

Variables $d_{o,j}(t), \alpha_{o,j}(t), \beta_{o,j}(t), d_v(t), d_a(t), \alpha_v(t), \alpha_a(t), \beta_a(t)$ and $\beta_v(t)$ are additional variables which will be computed along the projection part.

Reformulated Problem: Now, leveraging the above reformulations, we can re-phrase the projection problem (8a)-(8c) for the i^{th} sample (see Alg.1) as:

$$\bar{\boldsymbol{\xi}}_i = \arg\min_{\bar{\boldsymbol{\xi}}_i} \frac{1}{2}\|\bar{\boldsymbol{\xi}}_i - \boldsymbol{\xi}_i\|_2^2 \tag{13a}$$

$$\mathbf{A}\bar{\boldsymbol{\xi}}_i = \mathbf{b}_{eq}, \tag{13b}$$

$$\tilde{\mathbf{F}}\bar{\boldsymbol{\xi}}_i = \tilde{\mathbf{e}}(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i, \mathbf{d}_i), \tag{13c}$$

$$\mathbf{d}_{min} \leq \mathbf{d}_i \leq \mathbf{d}_{max}, \tag{13d}$$

$$\mathbf{G}\bar{\boldsymbol{\xi}}_i \leq \boldsymbol{\tau} \tag{13e}$$

where $\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i$ and $\mathbf{d}_i$ are the representation of $\begin{bmatrix}\boldsymbol{\alpha}_{o,i}^T & \boldsymbol{\alpha}_{v,i}^T & \boldsymbol{\alpha}_{a,i}^T\end{bmatrix}^T$, $\begin{bmatrix}\boldsymbol{\beta}_{o,i}^T & \boldsymbol{\beta}_{v,i}^T & \boldsymbol{\beta}_{a,i}^T\end{bmatrix}^T$, and $\begin{bmatrix}\mathbf{d}_{o,i}^T & \mathbf{d}_{v,i}^T & \mathbf{d}_{a,i}^T\end{bmatrix}^T$ respectively. The constant vector $\boldsymbol{\tau}$ is formed by stacking the s_{min} and s_{max} in appropriate form. The matrix $\mathbf{G}$ is formed by stacking $-\mathbf{P}$ and $\mathbf{P}$ vertically. Similarly, $\mathbf{d}_{min}$, $\mathbf{d}_{max}$ are formed by stacking the lower ($[1, 0, 0]$), and upper bounds ($[\infty, 1, 1]$) of $\mathbf{d}_{o,i}, \mathbf{d}_{v,i}, \mathbf{d}_{a,i}$. Also, $\tilde{\mathbf{F}}$, and $\mathbf{e}$ are formed as

$$\tilde{\mathbf{F}}=\begin{bmatrix} \begin{bmatrix}\mathbf{F}_o \\ \dot{\mathbf{P}} \\ \ddot{\mathbf{P}}\end{bmatrix} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \begin{bmatrix}\mathbf{F}_o \\ \dot{\mathbf{P}} \\ \ddot{\mathbf{P}}\end{bmatrix} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \begin{bmatrix}\mathbf{F}_o \\ \dot{\mathbf{P}} \\ \ddot{\mathbf{P}}\end{bmatrix} \end{bmatrix}, \tilde{\mathbf{e}}=\begin{bmatrix} \mathbf{x}_o + a\mathbf{d}_{o,i}\cos\boldsymbol{\alpha}_{o,i}\sin\boldsymbol{\beta}_{o,i} \\ \mathbf{d}_{v,i}v_{max}\cos\boldsymbol{\alpha}_{v,i}\sin\boldsymbol{\beta}_{v,i} \\ \mathbf{d}_{a,i}a_{max}\cos\boldsymbol{\alpha}_{a,i}\sin\boldsymbol{\beta}_{a,i} \\ \mathbf{y}_o + a\mathbf{d}_{o,i}\sin\boldsymbol{\alpha}_{o,i}\sin\boldsymbol{\beta}_{o,i} \\ \mathbf{d}_{v,i}v_{max}\sin\boldsymbol{\alpha}_{v,i}\sin\boldsymbol{\beta}_{v,i} \\ \mathbf{d}_{a,i}a_{max}\sin\boldsymbol{\alpha}_{a,i}\sin\boldsymbol{\beta}_{a,i} \\ \mathbf{z}_o + b\,\mathbf{d}_{o,i}\cos\boldsymbol{\beta}_{o,i} \\ \mathbf{d}_{v,i}v_{max}\cos\boldsymbol{\beta}_{v,i} \\ \mathbf{d}_{a,i}a_{max}\cos\boldsymbol{\beta}_{a,i} \end{bmatrix}, \tag{14}$$

where $\mathbf{F}_o$ is formed by stacking as many times as the number of obstacles. Also, $\mathbf{x}_o, \mathbf{y}_o, \mathbf{z}_o$ is obtained by stacking $x_{o,j}(t), y_{o,j}(t), z_{o,j}(t)$ at different time stamps and for all obstacles.

Solution process We utilize the augmented Lagrangian method to relax the equality and affine constraints (13c)-(13e) as l_2 penalties. Consequently, the projection cost can be rephrased as follows:

$$\mathcal{L}=\frac{1}{2}\left\|\overline{\boldsymbol{\xi}}_i-\boldsymbol{\xi}_i\right\|_2^2-\langle\boldsymbol{\lambda}_i,\overline{\boldsymbol{\xi}}_i\rangle+\frac{\rho}{2}\left\|\tilde{\mathbf{F}}\overline{\boldsymbol{\xi}}_i-\tilde{\mathbf{e}}\right\|_2^2+\frac{\rho}{2}\left\|\mathbf{G}\overline{\boldsymbol{\xi}}_i-\boldsymbol{\tau}+\mathbf{s}_i\right\|_2^2,$$
$$=\frac{1}{2}\left\|\overline{\boldsymbol{\xi}}_i-\boldsymbol{\xi}_i\right\|_2^2-\langle\boldsymbol{\lambda}_i,\overline{\boldsymbol{\xi}}_i\rangle+\frac{\rho}{2}\left\|\mathbf{F}\overline{\boldsymbol{\xi}}_i-\mathbf{e}\right\|_2^2 \quad (15)$$

where, $\mathbf{F}=\begin{bmatrix}\tilde{\mathbf{F}}\\ \mathbf{G}\end{bmatrix}$, $\mathbf{e}=\begin{bmatrix}\tilde{\mathbf{e}}\\ \boldsymbol{\tau}-\mathbf{s}_i\end{bmatrix}$. Also, $\boldsymbol{\lambda}_i$, ρ and $\mathbf{s}_i$ are Lagrange multiplier, scalar constant and slack variable. We minimize (15) subject to (13b) using AM, which is reduced to the following steps.

$$^{k+1}\boldsymbol{\alpha}_i=\arg\min_{\boldsymbol{\alpha}_i}\mathcal{L}(^k\overline{\boldsymbol{\xi}}_i,\boldsymbol{\alpha}_i,\ ^k\boldsymbol{\beta}_i,\ ^k\mathbf{d}_i,\ ^k\boldsymbol{\lambda}_i,\ ^k\mathbf{s}_i) \quad (16a)$$

$$^{k+1}\boldsymbol{\beta}_i=\arg\min_{\boldsymbol{\beta}_i}\mathcal{L}(^k\overline{\boldsymbol{\xi}}_i,\ ^{k+1}\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i,\ ^k\mathbf{d}_i,\ ^k\boldsymbol{\lambda}_i,\ ^k\mathbf{s}_i) \quad (16b)$$

$$^{k+1}\mathbf{d}_i=\arg\min_{\mathbf{d}_i}\mathcal{L}(\ ^k\overline{\boldsymbol{\xi}}_i,\ ^{k+1}\boldsymbol{\alpha}_i,\ ^{k+1}\boldsymbol{\beta}_i,\mathbf{d}_i,\ ^k\boldsymbol{\lambda}_i,\ ^k\mathbf{s}_i) \quad (16c)$$

$$^{k+1}\mathbf{s}_i=\max(\mathbf{0},-\mathbf{G}\ ^k\overline{\boldsymbol{\xi}}_i+\boldsymbol{\tau}) \quad (16d)$$

$$^{k+1}\boldsymbol{\lambda}_i=^k\boldsymbol{\lambda}_i-\rho\mathbf{F}^T(\mathbf{F}\ ^k\overline{\boldsymbol{\xi}}_i-^k\tilde{\mathbf{e}}) \quad (16e)$$

$$^{k+1}\mathbf{e}=\begin{bmatrix}\tilde{\mathbf{e}}(\ ^{k+1}\boldsymbol{\alpha}_i,^{k+1}\boldsymbol{\beta}_i,^{k+1}\mathbf{d}_i)\\ \boldsymbol{\tau}-^{k+1}\mathbf{s}_i\end{bmatrix} \quad (16f)$$

$$^{k+1}\overline{\boldsymbol{\xi}}_i=\arg\min_{\overline{\boldsymbol{\xi}}_i}\mathcal{L}(\boldsymbol{\xi}_i,\ ^{k+1}\mathbf{e}_i,^{k+1}\boldsymbol{\lambda}_i,) \quad (16g)$$

For each AM step, we only optimize one group of variables while others are held fixed. Note that stacking of right-hand sides of (16f) and (16e) provide the function **h** presented in (6a). The steps (16a)-(16c) have closed form solutions in terms of $^k\overline{\boldsymbol{\xi}}_i$ [27], [13], [21]. Also, (16g) is a representation of (8a)-(8c). Our projection optimizer is inspired by the algorithm presented in [29] for training deep neural networks based on AM approach. Although, a general convergence proof is beyond the scope of this work, we present empirical convergence validation plots in the accompanying video.

Remark 5. *The matrix* $\boldsymbol{F}$ *and vector* $\boldsymbol{e}$ *(see* (14)*) have the dimension of* $3(n_o+2)n_p\times 3n_v$ *and* $3(n_0+2)n_p$ *and their complexity grow linearly with the number of obstacles* n_o *and the planning horizon* n_p*.*

REFERENCES

[1] C. Rösmann, F. Hoffmann, and T. Bertram, "Kinodynamic trajectory optimization and control for car-like robots," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 5681–5686.

[2] C. Metz, "Rockit software," *http://xray. bsd. uchicago. edu/krl/KRL_ROC/software_index6. htm*, 2003.

[3] L. Vanroye, A. Sathya, J. De Schutter, and W. Decré, "Fatrop: A fast constrained optimal control problem solver for robot trajectory optimization and control," *arXiv preprint arXiv:2303.16746*, 2023, to appear at IROS 2023.

[4] H. Bharadhwaj, K. Xie, and F. Shkurti, "Model-predictive control via cross-entropy and gradient-based optimization," in *Learning for Dynamics and Control*. PMLR, 2020, pp. 277–286.

[5] J. Jankowski, L. Brudermüller, N. Hawes, and S. Calinon, "Vp-sto: Via-point-based stochastic trajectory optimization for reactive robot behavior," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 10 125–10 131.

[6] L. Petrović, J. Peršić, M. Seder, and I. Marković, "Cross-entropy based stochastic optimization of robot trajectories using heteroscedastic continuous-time gaussian processes," *Robotics and Autonomous Systems*, vol. 133, p. 103618, 2020.

[7] M.-G. Kim and K.-K. K. Kim, "Mppi-ipddp: Hybrid method of collision-free smooth trajectory generation for autonomous robots," *arXiv preprint arXiv:2208.02439*, 2022.

[8] Z. Zhang, J. Jin, M. Jagersand, J. Luo, and D. Schuurmans, "A simple decentralized cross-entropy method," *Advances in Neural Information Processing Systems*, vol. 35, pp. 36 495–36 506, 2022.

[9] D. Perille, A. Truong, X. Xiao, and P. Stone, "Benchmarking metric ground navigation," in *2020 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, 2020.

[10] 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.

[11] 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.

[12] Z. Han, Y. Wu, T. Li, L. Zhang, L. Pei, L. Xu, C. Li, C. Ma, C. Xu, S. Shen *et al.*, "An efficient spatial-temporal trajectory planner for autonomous vehicles in unstructured environments," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[13] F. Rastgar, H. Masnavi, J. Shrestha, K. Kruusamäe, A. Aabloo, and A. K. Singh, "Gpu accelerated convex approximations for fast multi-agent trajectory optimization," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3303–3310, 2021.

[14] D. Guhathakurta, F. Rastgar, M. A. Sharma, K. M. Krishna, and A. K. Singh, "Fast joint multi-robot trajectory optimization by gpu accelerated batch solution of distributed sub-problems," *Frontiers in Robotics and AI*, vol. 9, p. 890385, 2022.

[15] K. Fatahalian, J. Sugerman, and P. Hanrahan, "Understanding the efficiency of gpu algorithms for matrix-matrix multiplication," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, 2004, pp. 133–137.

[16] M. Bhardwaj, B. Sundaralingam, A. Mousavian, N. D. Ratliff, D. Fox, F. Ramos, and B. Boots, "Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation," in *Conference on Robot Learning*. PMLR, 2022, pp. 750–759.

[17] M. Wen and U. Topcu, "Constrained cross-entropy method for safe reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[18] 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.

[19] 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.

[20] D. M. Asmar, R. Senanayake, S. Manuel, and M. J. Kochenderfer, "Model predictive optimized path integral strategies," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 3182–3188.

[21] H. Masnavi, J. Shrestha, M. Mishra, P. Sujit, K. Kruusamäe, and A. K. Singh, "Visibility-aware navigation with batch projection augmented cross-entropy method over a learned occlusion cost," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9366–9373, 2022.

[22] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/google/jax

[23] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[24] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3d: A modern library for 3d data processing," *arXiv preprint arXiv:1801.09847*, 2018.

[25] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[26] I. S. Mohamed, M. Ali, and L. Liu, "Gp-guided mppi for efficient navigation in complex unknown cluttered environments," *arXiv preprint arXiv:2307.04019, to appear in IROS 2023*, 2023.

[27] F. Rastgar, A. K. Singh, H. Masnavi, K. Kruusamae, and A. Aabloo, "A novel trajectory optimization for affine systems: Beyond convex-concave procedure," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 1308–1315.

[28] F. Rastgar, H. Masnavi, K. Kruusamäe, A. Aabloo, and A. K. Singh, "Gpu accelerated batch trajectory optimization for autonomous navigation," in *2023 American Control Conference (ACC)*. IEEE, 2023, pp. 718–725.

[29] G. Taylor, R. Burmeister, Z. Xu, B. Singh, A. Patel, and T. Goldstein, "Training neural networks without gradients: A scalable admm approach," in *International conference on machine learning*. PMLR, 2016, pp. 2722–2731.