BOW: Bayesian Optimization over Windows for Motion Planning in Complex Environments

Sourav Raxit[1], Abdullah Al Redwan Newaz[1*], Paulo Padrao[3], Jose Fuentes[2], and Leonardo Bobadilla[2]

Abstract—**This paper introduces the BOW Planner, a scalable motion planning algorithm designed to navigate robots through complex environments using constrained Bayesian optimization (CBO). Unlike traditional methods, which often struggle with kinodynamic constraints such as velocity and acceleration limits, the BOW Planner excels by concentrating on a planning window of reachable velocities and employing CBO to sample control inputs efficiently. This approach enables the planner to manage high-dimensional objective functions and stringent safety constraints with minimal sampling, ensuring rapid and secure trajectory generation. Theoretical analysis confirms the algorithm's asymptotic convergence to near-optimal solutions, while extensive evaluations in cluttered and constrained settings reveal substantial improvements in computation times, trajectory lengths, and solution times compared to existing techniques. Successfully deployed across various real-world robotic systems, the BOW Planner demonstrates its practical significance through exceptional sample efficiency, safety-aware optimization, and rapid planning capabilities, making it a valuable tool for advancing robotic applications. The BOW Planner is released as an open-source package and videos of real-world and simulated experiments are available at https://bow-web.github.io/.**

Index Terms—**Motion and Path Planning; Collision Avoidance; Nonholonomic Motion Planning**

I. INTRODUCTION

MOTION planning for mobile robots in complex environments remains a challenging problem in robotics. The task can be formulated as a constrained optimization problem, where the objective is to minimize a cost function (e.g., time, energy) to reach a goal destination, subject to constraints such as obstacle avoidance and kinematic/dynamic feasibility [1, 2]. However, evaluating this constrained objective function over a long planning horizon is computationally expensive, especially for systems with complex dynamics and constraints [2].

To address this issue, the Receding Horizon Controller (RHC) approach solves the optimization problem over a shorter, fixed horizon and the solution is applied for a single step before re-optimizing. Early work in RHC highlighted the critical role of constraints in control processes by meticulously developing a model of the system [3]. Later, a unified motion planning and control solution was introduced, capable of navigating environments with both static and dynamic obstacles [4]. The Dynamic Window Approach (DWA) [5] searches robot control commands directly in velocity space, constraining the search to dynamically reachable velocities over short temporal windows. The Hybrid Reciprocal Velocity Obstacle (HRVO) [6] method complements this framework by providing closed-form characterization of collision-free velocities. HRVO shifts the velocity obstacle cone to account for mutual agent cooperation, enabling smooth, decentralized avoidance.

Recent advancements in motion planning include learning-based receding horizon planning [7], risk-aware planning [8], and perception-aware planning [9] to enhance performance within computational constraints. Deep learning-based motion planners fall into two categories: (i) end-to-end collision-free path generation and (ii) subsystem improvement methods [10]. End-to-end approaches utilize 3D CNNs, Point-Net++, and Transformers for robot locomotion and visuomotor control [11, 12, 13]. Subsystem frameworks like Dynamic MPNets [14] combine neural planning with model predictive control for robust solutions in learned environments.

In motion planning, Bayesian optimization (BO) plays a crucial role in identifying safe and efficient paths by strategically reducing uncertainty within the search space. This is achieved through the combination of BO with Gaussian Processes (GP), which are widely used in high-dimensional motion planning to optimize expensive-to-evaluate black-box functions [15]. BO employs GP as a surrogate model to iteratively sample the objective function, construct a probabilistic model, and use an acquisition function to identify the next most promising point for evaluation [16, 17]. Spatial GPs address the curse of dimensionality by selecting a small, spatially diverse subset of the dynamic window, thereby significantly reducing the sample space [18, 19], and they avoid overfitting by considering multiple potential curves for the underlying process. Additionally, GP-based motion planning frameworks, such as GPMP and GPMP2, leverage sparse GP representations to model continuous-time trajectories and perform efficient probabilistic inference via factor graphs, enabling fast optimization under kinodynamic constraints [20]. While these techniques perform well for unconstrained objectives, most motion planning problems require faster computation and multiple trajectory options under kinodynamic constraints. To address these challenges, recent methods integrate GP models into model predictive

[1] S. Raxit, and A. A. R. Newaz are with the Department of Computer Science, University of New Orleans, New Orleans, LA 70148, USA (email: {sraxit, aredwann}@uno.edu). [2] J. Fuentes, and L. Bobadilla are with the Knight Foundation School of Computing and Information Sciences, Florida International University, Miami, FL 33199, USA (email:{jfuen099@, bobadilla@cs.}fiu.edu). [3] P. Padrao is with Providence College, Department of Mathematics & Computer Science, Providence, RI 02918, USA (email:ppadraol@providence.edu).

control (MPC) frameworks to capture unmodeled dynamics and propagate uncertainty for safe control [21], while complementary approaches use Bayesian optimization to tune safety margins within MPC under uncertainty, effectively combining real-time constrained optimization with data-efficient learning for robust trajectory planning [22].

In this paper, we propose Bayesian Optimization over Windows (BOW), which incorporates the principles of the Dynamic Window Approach (DWA) [5] into the Constrained Bayesian Optimization (CBO) framework [23, 16]. Our approach fixes the control input over a short prediction horizon, serving two primary purposes: (i) ensuring computational tractability of the CBO-based optimization, and (ii) aligning with proven motion planning strategies that leverage short-horizon control to balance performance and feasibility. Unlike earlier methods that rely on heuristics or predefined sampling strategies, the BOW Planner's use of the Constrained Expected Improvement (CEI) acquisition function allows it to learn constraints and objectives simultaneously.

The main contributions of this paper are: (i) **Enhanced Sample Efficiency:** BOW can find optimal control policies with fewer evaluations, providing feasible solutions in significantly less time compared to existing methods. (ii) **Safety-Integrated Optimization:** Safety constraints are directly incorporated into the optimization process, ensuring solutions are both optimal and feasible. (iii) **Rapid Planning & Scalability:** This method outperforms state-of-the-art planning algorithms and scales effectively to tackle high-dimensional motion planning, supporting complex robotic tasks with numerous optimization parameters. (iv) **Theoretical Guarantees:** Beyond thorough benchmarking in many different cluttered environments, BOW offers proven guarantees of asymptotic convergence and near-optimal solutions, ensuring reliable performance in demanding settings.

II. PROBLEM FORMULATION

Consider a mobile robot with kinodynamic model operating in a d-dimensional environment $\mathcal{W} \subseteq \mathbb{R}^d$ containing K unknown obstacles. The robot's state space is denoted as $\mathcal{X} \subseteq \mathbb{R}^n$, and its control space is represented by $\mathcal{U} \subseteq \mathbb{R}^m$.

The robot's motion model describes how its state evolves over time based on the applied control inputs. Given the robot's current state $\mathbf{x}_t \in \mathcal{X}$ at time t, and a control input $\mathbf{u}_t \in \mathcal{U}$, the robot's next state $\mathbf{x}_{t+1}$ is computed using the following:

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

where $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is the state transition function.

The objective of the motion planning problem is to find a collision-free trajectory $\mathbf{x} : \{0, \ldots, T\} \to \mathcal{X}_{\text{free}}$ that takes the robot from its initial state $\mathbf{x}_{\text{init}} \in \mathcal{X}$ to a desired goal state $\mathbf{x}_{\text{goal}} \in \mathcal{X}_{\text{goal}} \subset \mathcal{X}$, while satisfying the motion model constraints. Here, $\mathcal{X}_{\text{free}} \subseteq \mathcal{X}$ represents the obstacle-free region of the state space, and T is the trajectory duration. Formally, the motion planning problem can be stated as:

$$\begin{aligned} \text{Find} \quad & \mathbf{x} : \{0, \ldots, T\} \to \mathcal{X}_{\text{free}} \\ \text{s.t.} \quad & \mathbf{x}_0 = \mathbf{x}_{\text{init}}, \, \mathbf{x}_T = \mathbf{x}_{\text{goal}}, \, \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t). \end{aligned} \tag{2}$$

To solve the motion planning problem, we can formulate it as an optimization problem by defining a cost-to-go function $J : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ that measures the quality of the robot's state and control inputs. The cost-to-go function can incorporate various objectives, such as minimizing the distance to the goal state, avoiding obstacles, or optimizing energy consumption. Additionally, we can impose constraints $c_k : \mathcal{X} \to \mathbb{R}$ to ensure the robot's safety and feasibility. The optimization problem can be expressed as:

$$\begin{aligned} \min_{\mathbf{u}_0, \ldots, \mathbf{u}_{T-1}} \quad & \sum_{t=0}^{T-1} J(\mathbf{x}_t, \mathbf{u}_t) + J_F(\mathbf{x}_T) \\ \text{s.t.} \quad & \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t), \quad t \in \{0, \ldots, T-1\}, \\ & c_k(\mathbf{x}_t) \leq 0, \quad k \in \{1, \ldots, K\}, \, t \in \{1, \ldots, T\}, \end{aligned} \tag{3}$$

where K is the number of constraints.

In the context of online motion planning, the robot has a limited time budget Δ to compute its control inputs at each time step. The robot must optimize its trajectory by solving the optimization problem in (3) within the allocated time budget. The recommended control input is applied to the robot, and the process repeats at the next timestep.

III. BOW PLANNER

Given a dynamic planning window $V_d \subset \mathcal{U}$ based on the robot's current velocity and acceleration limits [5], we need to evaluate the objective function J and constraints c_k for each control input $\mathbf{u} \in V_d$ by simulating the robot's trajectory over a short time interval $[t, t+\Delta) \subseteq [0, T]$.

a) Reachability Objective: To navigate to a desired goal state, we define the cost-to-go function by specifying an appropriate cost on the state and control, and solving for the optimal cost-to-go using the following function:

$$J(\mathbf{x}_t, \mathbf{u}_t) = \|\mathbf{x}_{\text{goal}} - f(\mathbf{x}_t, \mathbf{u}_t)\| = \|\mathbf{x}_{\text{goal}} - \mathbf{x}_{t+1}\|, \tag{4}$$

where $\|\cdot\|$ is Euclidean norm between a predicted state $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$ and the goal state $\mathbf{x}_{\text{goal}}$.

To reduce the computational effort in evaluating the cost-to-go function over a short-horizon trajectory, we consider a subset of possible trajectories where the same control is applied, i.e., $\mathbf{u}_\tau = \mathbf{u}_t$ for $\tau = t, \ldots, t+\Delta$. The future cost incurred by being in a given state with Δ steps to go is then

$$\mathcal{J}(\mathbf{x}_{[t:t+\Delta+1]}, \mathbf{u}_{[t:t+\Delta]}) = \sum_{\tau=t}^{t+\Delta} J(\mathbf{x}_\tau, \mathbf{u}_t) + J_F(\mathbf{x}_{t+\Delta+1}), \tag{5}$$

where $J(\mathbf{x}_\tau, \mathbf{u}_t)$ is the stage cost and $J_F(\mathbf{x}_{t+\Delta+1})$ is the terminal cost. This serves as a proxy for the original cost function in the optimization problem (3), reducing the computational cost of evaluating the entire path.

Furthermore, using (1), $\mathbf{x}_t$, and $\mathbf{u}_t$ to rewrite $\{\mathbf{x}_{t+\tau}$ for $\tau = 1, \ldots, \Delta\}$, it is possible to show that $\mathcal{J}$ depends only on $\mathbf{x}_t$ and $\mathbf{u}_t$ (with J, J_F, f, and Δ fixed). For that reason, we will use the shorthand $\mathcal{J}(\mathbf{x}_{[t:t+\Delta+1]}, \mathbf{u}_{[t:t+\Delta]}) \coloneqq \mathcal{J}(\mathbf{x}_t, \mathbf{u}_t)$ from now on.

Note that like most local planners [5, 6, 24, 25], we assume the cost-to-go function in Eqn. (5) is admissible for a given planning horizon.

b) Safety Constraints: In the context of collision avoidance, the constraint functions c_k can be defined to ensure that the robot does not collide with any obstacles. Let $\mathcal{O}_k$ represent the k-th obstacle in the environment. The collision constraint is defined as:

$$c_k(\mathbf{x}_t) = r_{\text{safe}} - d(\mathbf{x}_t, \mathcal{O}_k), \tag{6}$$

where $d(\mathbf{x}_t, \mathcal{O}_k)$ is the distance between the robot's state $\mathbf{x}_t$ and the obstacle $\mathcal{O}_k$, and r_{safe} is a safety radius that ensures a buffer zone around the obstacle to prevent collisions.

c) Approximating the Objective and Constraint Functions: We recall that at each timestep t, the function described in Equation (5) is optimized to find the next optimal control $\mathbf{u}_t^\star$. Since $\mathcal{J}$ may become expensive to compute, we utilize BO to guide this search efficiently. Let $\mathcal{J}(\mathbf{u}) = \mathcal{J}(\mathbf{x}_t, \mathbf{u})$, and $c_k(\mathbf{u}) = c_k(f(\mathbf{x}_t, \mathbf{u}))$. To compute $\mathbf{u}_t^\star$, BO models the objective function and the constraints as GPs over the domain V_d. We train the GPs by utilizing p evaluations of the objective function and constraints within V_d, constructing $K+1$ datasets:

$$
\begin{aligned}
D_{\mathcal{J}} &= \left\{ \left(\mathbf{u}_t^{(i)}, \mathcal{J}(\mathbf{x}_t, \mathbf{u}_t^{(i)}) \right) \right\}_{i=1}^p, \\
D_{c_k} &= \left\{ \left(\mathbf{u}_t^{(i)}, c_k(f(\mathbf{x}_t, \mathbf{u}_t^{(i)})) \right) \right\}_{i=1}^p, \; k = 1, \ldots, K.
\end{aligned}
\tag{7}
$$

Thus, the objective function and the constraints are approximated as GP realizations:

$$
\begin{aligned}
\mathcal{J}(\mathbf{u}) &\sim \mathcal{GP}(\mu_{\mathcal{J}}(\mathbf{u}), \mathcal{K}_{\mathcal{J}}(\mathbf{u}, \mathbf{u}')) \\
c_k(\mathbf{u}) &\sim \mathcal{GP}(\mu_{c_k}(\mathbf{u}), \mathcal{K}_{c_k}(\mathbf{u}, \mathbf{u}')), \; k = 1, \ldots, K.
\end{aligned}
\tag{8}
$$

Aside from approximating the objective function using GPs, BO samples promising points using an acquisition function that quantifies the probability of each sample to have a high value when evaluated by the objective function. In unconstrained scenarios, the standard Expected Improvement acquisition function $\text{EI}(\mathbf{u})$ is used as:

$$\text{EI}(\mathbf{u}) = (y^- - \mu_{\mathcal{J}}(\mathbf{u}))\Phi(z) + \sigma_{\mathcal{J}}(\mathbf{u})\phi(z), \tag{9}$$

where y^- is the current best observed objective value; $\mu_{\mathcal{J}}(\mathbf{u})$ is the posterior mean at $\mathbf{u}$; $\sigma_{\mathcal{J}}(\mathbf{u}) = \sqrt{\mathcal{K}_{\mathcal{J}}(\mathbf{u}, \mathbf{u})}$ is the posterior standard deviation at $\mathbf{u}$; $z = \frac{y^- - \mu_{\mathcal{J}}(\mathbf{u})}{\sigma_{\mathcal{J}}(\mathbf{u})}$. Φ and ϕ are the CDF and the PDF of the standard normal distribution, respectively. This function provides a reasonable trade-off between exploration and exploitation.

Building upon the Constrained Expected Improvement (CEI) acquisition function [26], we developed a modified EI that incorporates control input constraints to achieve a balance between expected objective function improvement and constraint satisfaction probability as:

$$\text{CEI}(\mathbf{u}) = \text{EI}(\mathbf{u})P(\text{feasible} \mid \mathbf{u}), \tag{10}$$

where

$$P(\text{feasible} \mid \mathbf{u}) = \prod_{k=1}^K P(c_k(\mathbf{u}) \leq 0 \mid D_{c_k}) \tag{11}$$

is the probability that all constraints are satisfied at $\mathbf{u}$ assuming the constraints are conditionally independent given $\mathbf{u}$.

Furthermore, each of these probabilities has a closed form as

$$P(c_k(\mathbf{u}) \leq 0 \mid D_{c_k}) = \Phi\left(\frac{-\mu_{c_k}(\mathbf{u})}{\sigma_{c_k}(\mathbf{u})} \right), \tag{12}$$

where $\mu_{c_k}(\mathbf{u})$ and $\sigma_{c_k}(\mathbf{u})$ are the posterior mean and standard deviation of the GP for the k-th constraint at $\mathbf{u}$.

Finally, the optimal control input $\mathbf{u}_t^\star$ to evaluate is chosen by maximizing the CEI:

$$\mathbf{u}_t^\star = \arg\max_{\mathbf{u} \in V_d} \text{CEI}(\mathbf{u}). \tag{13}$$

Equation (13) ensures that the optimization process not only seeks to improve the objective function but also respects the constraints by considering the probability of feasibility.

A. Receding Horizon Planning and Control

We employ a receding horizon optimization setup, where control actions are determined iteratively to guide the robot toward its goal. Algorithm 1 details the BOW planning procedures. The process begins by initializing an empty final trajectory, which will store the sequence of feasible states the robot follows. The algorithm then enters a continuous loop that persists until the robot reaches the goal, defined by a specified goal radius r.

Algorithm 1 BOW Planning and Control

1: **Input:** Initial state $\mathbf{x}_0$, Goal state $\mathbf{x}_{\text{goal}}$, Parameters ξ, Goal distance r
2: **Output:** Final trajectory $\mathbf{x}_{[0:t]}$, Optimal controls $\mathbf{u}_{[0:t-1]}^\star$
3: Initialize $t \leftarrow 0$, $\mathbf{x}_t \leftarrow \mathbf{x}_0$
4: **while** $\|\mathbf{x}_t - \mathbf{x}_{\text{goal}}\| \leq r$ **do**
5:   Define planning window $V_d \subseteq \mathcal{U}$ using f and ξ
6:   Sample p controls $\{\mathbf{u}_t^{(i)}\}_{i=1}^p \subseteq V_d$
7:   Initialize datasets: $D_{\mathcal{J}} \leftarrow \emptyset$ and $D_{c_k} \leftarrow \emptyset$, for $k = 1, \ldots, K$.
8:   **for** $i = 1, \ldots, p$ **do**
9:     Compute $\mathcal{J}(\mathbf{x}_t, \mathbf{u}_t^{(i)})$ using Equation (5)
10:     Compute constraint values $c_k(f(\mathbf{x}_t, \mathbf{u}_t^{(i)}))$ for each $k = 1, \ldots, K$ using Eqaution (6)
11:     Update datasets: $D_{\mathcal{J}} \leftarrow D_{\mathcal{J}} \cup \{(\mathbf{u}_t^{(i)}, \mathcal{J}(\mathbf{x}_t, \mathbf{u}_t^{(i)}))\}$ and $D_{c_k} \leftarrow D_{c_k} \cup \{\mathbf{u}_t^{(i)}, c_k(f(\mathbf{x}_t, \mathbf{u}_t^{(i)}))\}$
12:   **end for**
13:   Train the $K + 1$ GPs using $D_{\mathcal{J}}$ and D_{c_k}
14:   Compute $\text{CEI}(\mathbf{u})$ using Equations (9),(10) and (12)
15:   Choose $\mathbf{u}_t^\star = \arg\max_{\mathbf{u} \in V_d} \text{CEI}(\mathbf{u})$
16:   Generate trajectory $\hat{\mathbf{x}}_{[t:t+\Delta]} = (\hat{\mathbf{x}}_t, \hat{\mathbf{x}}_{t+1} \ldots, \mathbf{x}_{t+\Delta})$ using RK4 on f, $\mathbf{u}_t^\star$ and $\mathbf{x}_t = \hat{\mathbf{x}}_t$
17:   Apply control $\mathbf{u}_t^\star$ for $t + \ell$ and obtain trajectory $\hat{\mathbf{x}}_{[t:t+\ell]}$ whereas $\ell \leq \Delta$
18:   **if** $c_k(\mathbf{x}_\tau) \leq 0$ for $\tau = t, \ldots, t + \ell$, $k = 1, \ldots, K$ **then**
19:     $\mathbf{x}_t \leftarrow \hat{\mathbf{x}}_{t+\ell}$
20:     $t \leftarrow t + \ell$
21:   **end if**
22: **end while**
23: **return** $\mathbf{x}_{[0:t]}$, $\mathbf{u}_{[0:t]}^\star$

At each iteration (line 4), the BOW Planner is instantiated with the current state $\mathbf{x}_t$, the goal $\mathbf{x}_{\text{goal}}$, a collision checker,

and a set of parameters ξ. This planner leverages the CBO approach to compute the optimal control $\mathbf{u}_t^*$.

As shown in lines 5–15, p control candidates are sampled from a local planning window V_d, such that $\{\mathbf{u}_t^{(i)}\}_{i=1}^p \subseteq V_d$. The cost and constraint satisfaction of each control $\mathbf{u}_t^{(i)}$ are evaluated using Equations (5)–(6). The CEI acquisition function facilitates effective constraint handling by modeling both the objective and constraint functions via GPs, ensuring that the selected control satisfies feasibility conditions such as obstacle avoidance.

Using the optimal control $\mathbf{u}_t^*$, a receding horizon trajectory is generated via the motion model at line 16, using fourth-order Runge-Kutta (RK4) integration. This trajectory consists of a sequence of predicted states over the planning horizon. The algorithm then verifies whether this trajectory is collision-free using a collision checker.

If the trajectory is valid, the algorithm determines an index ℓ (line 17), which indicates how far into the planning horizon the same control $\mathbf{u}_t^\star$ can be applied. This allows the method to extend control application over multiple future steps, offering greater flexibility than traditional Model Predictive Control (MPC), which typically only applies the immediate control at $t+1$.

In practical implementation with finite samples, a single evaluation of the CEI in Eqn. (13) generates solutions that formulate collision avoidance as soft chance constraints, thereby identifying candidate points that simultaneously enhance the objective function while maintaining high probability of safety compliance. The BOW planner employs a trajectory validation mechanism at line 18, which evaluates the complete predicted trajectory $\mathbf{x}_\tau$ across the time horizon $\tau = t, \ldots, t+\ell$. Upon successful validation where the trajectory satisfies all imposed constraints c_k and maintains collision-free operation throughout the prediction horizon, the algorithm updates the current state $\mathbf{x}_t$ to the predicted state at index ℓ, effectively advancing the planning window. Conversely, when the predicted trajectory fails to meet the feasibility criteria, the planner initiates an iterative sampling process for alternative control inputs, continuing until either a feasible trajectory is identified or the predetermined maximum iteration threshold is exceeded.

The loop proceeds by evaluating the distance between the updated state $\mathbf{x}_t$ and the goal $\mathbf{x}_{\text{goal}}$. If this distance is less than or equal to the threshold r (line 20), the algorithm terminates, indicating that the robot has successfully reached its target. Upon termination, the algorithm returns a success flag and the final trajectory, representing the complete path from the initial state to the goal.

IV. Algorithm Analysis

Lemma 1. *Let $g : V_d \subseteq \mathcal{U} \to \mathbb{R}$ be a Lipschitz continuous function on a compact set V_d. If a GP with an universal kernel (e.g., squared exponential kernel, see [27]) models g, then the posterior mean $\mu(\mathbf{u})$ converges uniformly to $g(\mathbf{u})$ as the number of samples tends to infinity.*

Sketch of the Proof. [27] shows that the set of functions that the squared exponential kernel generates is dense in $C(V_d)$. Moreover, [28] shows that the error, expressed in some norm

defined on V_d, converges to zero when the number of samples tends to infinity.

$\square$

Theorem 1. *Let $V_d \subseteq \mathcal{U}$ be a compact planning window defined by the robot's velocity and acceleration limits. Suppose the objective function $\mathcal{J}(\mathbf{u})$ and constraint functions $c_k(\mathbf{u})$ for $k = 1, \ldots, K$ are Lipschitz continuous over V_d. Assume the system dynamics f are Lipschitz continuous, and the GP models $\mathcal{J}$ and c_k use kernels such as the squared exponential kernel. Then, as the number of samples tends to infinity, the control input $\mathbf{u}_t^\star$ selected by maximizing the CEI converges to the true optimal control input $\mathbf{u}_{t,opt} = \arg\min_{\mathbf{u} \in V_d} \mathcal{J}(\mathbf{u})$ within V_d while satisfying all constraints $c_k(\mathbf{u}) \le 0$.*

Proof. We prove this theorem in five steps, leveraging properties of GPs and BO under the given assumptions.

Step 1: GP Approximation and Uniform Convergence: Since $\mathcal{J}(\mathbf{u})$ and $c_k(\mathbf{u})$ are Lipschitz continuous on the compact set V_d, Lemma 1 guarantees that their GP models converge to the functions as the number of samples increases.

Step 2: Convergence of the Probability of Feasibility: For each constraint $c_k(\mathbf{u})$, the probability of feasibility is modeled as

$$P(c_k(\mathbf{u}) \le 0 \mid D_{c_k}) = \Phi\left(\frac{-\mu_{c_k}(\mathbf{u})}{\sigma_{c_k}(\mathbf{u})}\right), \tag{14}$$

recalling that Φ is the standard normal CDF, and D_{c_k} is the dataset of constraint observations. As the number of samples tends to infinity, $\mu_{c_k}(\mathbf{u}) \to c_k(\mathbf{u})$ and $\sigma_{c_k}(\mathbf{u}) \to 0$. Thus,

$$P(c_k(\mathbf{u}) \le 0 \mid D_{c_k}) \to \begin{cases} 1 & \text{if } c_k(\mathbf{u}) < 0, \\ 0.5 & \text{if } c_k(\mathbf{u}) = 0, \\ 0 & \text{if } c_k(\mathbf{u}) > 0. \end{cases}$$

In the feasible region where $c_k(\mathbf{u}) \le 0$ for all k (assuming strict feasibility $c_k(\mathbf{u}) < 0$ for simplicity), $P(\text{feasible} \mid \mathbf{u}) \to 1$ (recall Equation (11)). For infeasible $\mathbf{u}$, where $c_k(\mathbf{u}) > 0$ for some k, $P(\text{feasible} \mid \mathbf{u}) \to 0$.

Step 3: Convergence of the EI process: [29] shows that an EI process produces a dense set of samples that converges to $\arg\min_{\mathbf{u} \in V_d} \mathcal{J}(\mathbf{u})$ when the number of samples tends to infinity.

Step 4: Behavior of the CEI function: Recall that CEI combines EI with feasibility:

$$\text{CEI}(\mathbf{u}) = \text{EI}(\mathbf{u})P(\text{feasible} \mid \mathbf{u}). \tag{15}$$

As the number of samples tends to infinity, we combine the results of the two previous steps, then, the CEI acquisition function becomes

$$\text{CEI}(\mathbf{u}) = \text{EI}(\mathbf{u}) \prod_{k=1}^K \mathbf{1}_{\{c_k(\mathbf{u}) \le 0\}}. \tag{16}$$

Step 5: Convergence to the Optimal Solution: Direclty from Equation (16), we have that

$$\mathbf{u}_t^\star = \arg\max_{\mathbf{u} \in V_d} \text{CEI}(\mathbf{u}) = \arg\max_{\mathbf{u} \in V_d} \text{EI}(\mathbf{u}) \prod_{k=1}^K \mathbf{1}_{\{c_k(\mathbf{u}) \le 0\}}$$
$$= \arg\max_{\mathbf{u} \in V_d,\, c_k(\mathbf{u}) \le 0,\, \forall k} \mathcal{J}(\mathbf{u}) = \mathbf{u}_{\text{opt}}, \tag{17}$$

completing the proof. $\square$

A. Computational Complexity of BOW Planner

The BOW Planning algorithm iteratively computes an optimal control $\mathbf{u}_t^\star$ using BO with GPs to generate collision-free trajectories over ψ iterations. With a single optimization iteration ($\psi = 1$) and a constant number of GP samples p, the complexity per iteration includes:

- **Control Optimization**: GP update costs $O(p^3)$, a constant since p is fixed.
- **Trajectory Generation**: RK4 integration over horizon Δ with state dimension n costs $O(\Delta n)$.
- **Collision Checking**: Collision checking of Δ points using a Bounding Volume Hierarchy (BVH) model achieves a per-query complexity of $O(\log C)$, resulting in a total complexity of $O(\Delta \log C)$, where C represents the number of geometric primitives in the hierarchy.

Total complexity is:

$$O(\psi(p^3 + \Delta(n + \log C))) = O(\psi\Delta(n + \log C))$$

where ψ is the number of iterations (problem-dependent), Δ is the horizon length, n is the state dimension, and C is the collision checking complexity.

V. EXPERIMENTS AND RESULTS

All experiments were conducted on a desktop computer running Ubuntu 22.04 LTS, equipped with an AMD Ryzen™ 9 7950X processor, an NVIDIA GTX 4070 GPU, and 32 GB of RAM. Robot localization was performed using a VICON motion capture system operating at 120 Hz. The BOW Planner was configured with a maximum speed of 1.0 m/s, a minimum speed of 0.0 m/s, a maximum acceleration of 0.5 m/s², a maximum yaw rate of 0.6981 rad/s, and a maximum yaw acceleration of 2.0472 rad/s². The ℓ was set to 7. The planning operated with a time step of 0.1 s and a prediction horizon of 3.0 s. For trajectory optimization, we used a Squared Exponential ARD kernel (Radial Basis Function) within the Bayesian Optimization framework.

A. Sample Efficiency

The sample efficiency property of the BOW planner is further illustrated in Fig. 1. Given a dynamic window,
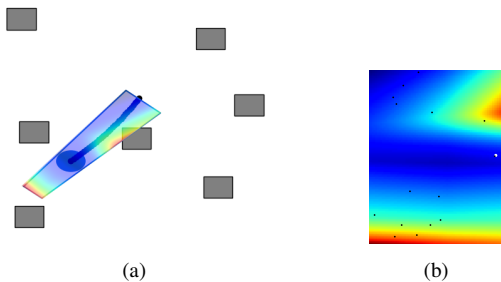

Fig. 1: **Sample Efficiency:** The DWA performs a grid search to assess the cost function (depicted by the colored map) using the dynamic window, while the BOW employs constrained Bayesian optimization to efficiently determine the optimal control (represented by white dots) with 15 samples only (indicated by black dots).

the DWA planner performs a grid search to compute the

optimal solution over the cost function, depicted as a colored map. In contrast, the MPPI planner uniformly samples the cost function from both feasible (blue-colored region) and infeasible (obstacle regions) spaces. Unlike other methods, the BOW planner efficiently computes the optimal solution (white dot) by evaluating samples from the feasible space more than the infeasible space, resulting in faster runtime efficiency. This demonstrates the superior sample efficiency of the BOW planner. Fig. 1 (b) illustrates the sampling behavior of the BOW planner. In this representation, the black dots indicate the samples taken by the BOW planner, while the white dot marks the optimal sample, and the colored map represents the cost map.

B. Unmanned Ground Vehicle Experiments

We implemented the BOW planner for a non-holonomic differential wheel UGV(Unmanned Ground Vehicle) navigating through a 2-D space containing obstacles. We performed simulations for UGVs across six different environments, as shown in Fig. 3. The UGV's state is defined by a 5-D vector $\mathbf{x} = (x, y, \theta, v, \omega)$, encompassing its position (x, y), orientation θ, linear velocity v and angular velocity ω. The control input is represented as a 2-D vector $\mathbf{u} = (v_c, \omega_c)$, which includes the linear velocity v_c and angular velocity ω_c with respect to the body frame.
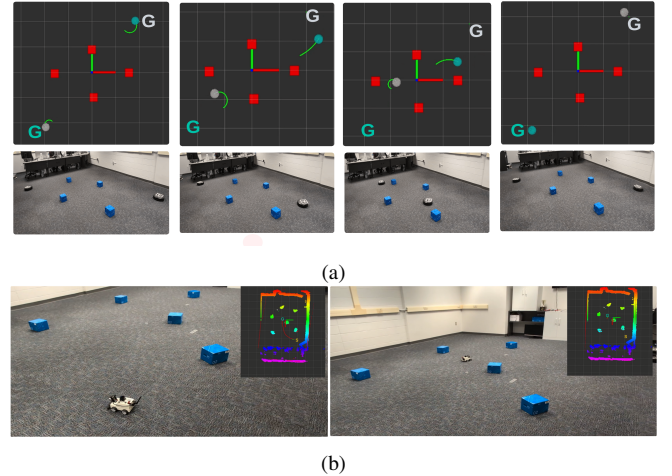

(a)

(b)

Fig. 2: The BOW planner is applied to UGVs for navigating environments with dynamic obstacles, leveraging its rapid planning capabilities 2a. The planner also has the ability to avoid obstacles using onboard LiDAR and to follow the planned trajectory 2b.

The faster approximation of optimal control enables the BOW planner to scale effectively for distributed robot navigation problems. Fig. 2a demonstrates this capability in a two-UGV scenario where each robot navigates toward its respective goal while treating the other as a dynamic obstacle. The gray UGV targets the gray "G" while the cyan UGV aims for the cyan "G". The color coding helps differentiate the robots' paths as they adjust their trajectories to avoid collisions during navigation. For this experiment, pose information was shared between robots, with the planner running on a desktop computer that sent commands to the UGVs. Fig. 2b illustrates the BOW planner's implementation on a unicycle UGV navigating an environment with static obstacles. The robot uses
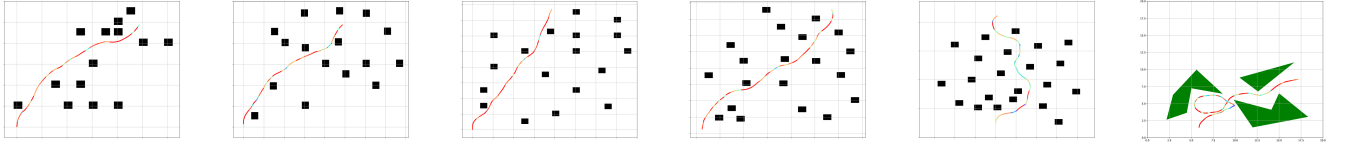
Fig. 3: Six simulated 20m×20m environments (Environments 1-6, from left to right) where black squares represent obstacles in the first five, and the sixth environment features non-convex obstacles composed of clustered green triangles. The curved trajectories display a blue-to-red gradient indicating increasing linear velocity from low to high.

an onboard laser scanner to detect obstacles and generate an occupancy map, enabling it to avoid obstacles while tracking the trajectory planned by the BOW planner. The UGV's goal location is indicated by the cyan "G", and the figure shows the planned path as the robot navigates around obstacles to reach its destination.

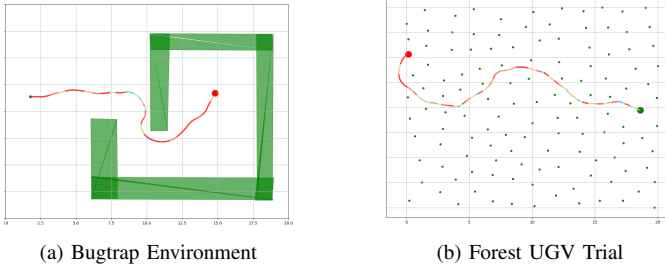(a) Bugtrap Environment (b) Forest UGV Trial

Fig. 4: The BOW planner is evaluated for UGV navigation in Bugtrap environments and Poisson forest environments featuring minimum obstacle density of 1.5 per m^2.

C. Benchmark

We evaluate the performance of the BOW planner against five established path planning algorithms: RRT [30], DWA [5], MPPI [24], HRVO [6], and CBF [25]. For each environment and planner combination, we conducted 10 independent runs and calculated the mean and standard deviation of all key metrics except Time/Step to ensure statistical reliability.

Evaluation across six distinct environments, as shown in Fig. 3, reveals that the BOW planner, using only 5 samples per planning horizon, consistently achieves the fastest total planning times —ranging from 10.00 ± 0.33 ms to 30.90 ± 0.68 ms, and per-step execution times between 0.10 and 0.15 ms, while maintaining competitive safe-trajectory lengths from 7.56 ± 0.19 m to 18.67 ± 0.12 m. In contrast, while MPPI produced the shortest paths in Envs. 1 and 3 with trajectories of 13.98 ± 0.00 m and 17.27 ± 0.00 m respectively, and CBF excelled in Env. 4 with 18.09 ± 0.00 m, both methods showed substantially higher computational demands with CBF requiring 9892.10 ± 97.26 ms to 15135.40 ± 335.45 ms total planning time and 63.41 to 88.51 ms per step, while MPPI demanded 5865.90 ± 182.73 ms to 44312.20 ± 645.95 ms total time and 42.28 to 43.81 ms per step, highlighting significant efficiency trade-offs. Critical completeness issues emerged across algorithms, with CBF failing to generate solutions in Envs. 2, 5, and 6, DWA unable to solve Env. 5, and HRVO failing in Env. 6, while BOW maintained perfect completeness across all test scenarios. RRT consistently required the fewest planning steps ranging from 31.7 ± 3.7 to 77.9 ± 10.0 but produced longer trajectories, while MPPI showed poor scalability, requiring up to 1048.0 ± 0.00 steps in Env. 6 compared to 137.0 ± 0.00 in simpler environments.

Additionally, velocity and smoothness characteristics varied significantly across planners, with MPPI achieving the highest average velocities in Envs. 1-4 ranging from 1.467 ± 0.026 to 1.774 ± 0.089 m/s, while BOW maintained consistent velocity performance between 0.939 ± 0.020 to 1.007 ± 0.077 m/s across all environments. In terms of trajectory smoothness, CBF achieved the lowest jerk values (-0.020 ± 0.001 to -0.002 ± 0.000), indicating very smooth paths, followed by DWA (0.028 ± 0.000 to 0.043 ± 0.000). BOW showed moderate smoothness (0.158 ± 0.029 to 0.357 ± 0.052), while MPPI exhibited highly variable performance (-2.647 ± 0.172 to 2.062 ± 0.291). RRT consistently had the highest jerk values (0.811 ± 0.060 to 0.927 ± 0.100), indicating the least smooth trajectories. These results show that the BOW planner is well-suited for resource-constrained, real-time applications requiring robust performance in diverse environments.

D. Unmanned Aerial Vehicle Experiments

To showcase the BOW planner's scalability in high-dimensional motion planning problems, we applied it to a 6-DOF aerial robot operating in a 3-D environment with 3-D obstacles (Fig. 5). The robot's state is represented by a 12-D vector $\mathbf{x} = (x, y, z, \theta, \dot{x}, \dot{y}, \dot{z}, \dot{\theta}, \ddot{x}, \ddot{y}, \ddot{z}, \ddot{\theta})$, encompassing position, orientation, velocity, and acceleration. The control input is a 4-D vector $\mathbf{u} = (\dot{x}_c, \dot{y}_c, \dot{z}_c, \dot{\theta}_c)$, consisting of linear velocities and yaw rate relative to the body frame. We evaluated the BOW planner using a quadrotor UAV in both simulations and real-world experiments. Fig. 5 depicts five distinct simulated environments, with red cubes representing obstacles. In each scenario, an Unmanned Aerial Vehicle (UAV) starts from an initial position and navigates to a goal location using the BOW planner. These simulations were conducted in the CoppeliaSim Simulator.

Fig. 6 illustrates the implementation of the BOW planner in a UAV's navigation through two different problems in terms of start and goal locations. Although the BOW planner is lightweight enough to run on edge devices like the Raspberry Pi or Jetson Nano, it was executed on a desktop due to the off-the-shelf UAV used, with control commands sent to the UAV during the experiments. In these scenarios, obstacles are represented by yellow cubes. The goal locations are marked with red labels "**G1**" and "**G2**", while the start locations are denoted by rose-colored labels "**S1**" and "**S2**". The figure shows the UAV's path as it successfully avoids obstacles to reach its destination.

Fig. 6a and Fig. 6b illustrate the UAV's generated path from start to goal location in two real environments with obstacles. These results demonstrate the UAV's obstacle avoidance capabilities and its ability to navigate to specific destinations in a 3D environment.

Env	Method	Lib + Lang	Steps	Traj. Length (m)	Total Time (ms)	Time / Step (ms)	Avg Velocity	Avg Jerk
1	BOW	FCL + C++	161.2 ± 2.9	14.14 ± 0.09	**23.26 ± 0.35**	**0.14**	0.947 ± 0.020	0.190 ± 0.024
	DWA	FCL + C++	157.0 ± 0.00	14.33 ± 0.00	103.68 ± 2.07	0.66	0.952 ± 0.00	0.031 ± 0.000
	HRVO	C++	539.0 ± 0.00	15.78 ± 0.00	269.22 ± 7.21	0.50	0.293 ± 0.00	0.223 ± 0.00
	RRT	FCL + C++	**55.2 ± 8.0**	17.39 ± 1.38	389.83 ± 272.41	7.06	0.796 ± 0.038	0.849 ± 0.062
	MPPI	CUDA + Py	137.0 ± 0.00	**13.98 ± 0.00**	5865.90 ± 182.73	42.82	**1.654 ± 0.052**	-1.150 ± 0.108
	CBF	Gurobi + Py	156.0 ± 0.00	14.42 ± 0.00	9892.10 ± 97.26	63.41	0.992 ± 0.010	**-0.020 ± 0.001**
2	BOW	FCL + C++	160.5 ± 3.2	**14.21 ± 0.08**	**23.09 ± 0.59**	**0.14**	0.955 ± 0.022	0.188 ± 0.026
	DWA	FCL + C++	332.0 ± 0.00	30.26 ± 0.00	83.15 ± 1.78	0.25	1.032 ± 0.000	**0.040 ± 0.000**
	HRVO	C++	341.0 ± 0.00	14.87 ± 0.00	282.89 ± 7.65	0.83	0.436 ± 0.00	0.137 ± 0.00
	RRT	FCL + C++	**57.0 ± 10.2**	17.30 ± 2.56	464.82 ± 87.55	8.15	0.765 ± 0.036	0.859 ± 0.085
	MPPI	CUDA + Py	156.0 ± 0.00	14.38 ± 0.00	6625.00 ± 146.45	42.47	**1.467 ± 0.032**	-2.647 ± 0.172
	CBF	Gurobi + Py	—	—	—	—	—	—
3	BOW	FCL + C++	199.2 ± 3.8	17.48 ± 0.06	**28.14 ± 0.58**	**0.14**	0.939 ± 0.020	0.190 ± 0.016
	DWA	FCL + C++	220.0 ± 0.00	19.14 ± 0.00	102.50 ± 1.83	0.47	0.889 ± 0.000	0.028 ± 0.00
	HRVO	C++	422.0 ± 0.00	20.07 ± 0.00	468.44 ± 10.51	1.11	0.476 ± 0.00	0.302 ± 0.000
	RRT	FCL + C++	**69.2 ± 12.7**	20.69 ± 2.63	808.09 ± 533.78	11.68	0.783 ± 0.022	0.886 ± 0.050
	MPPI	CUDA + Py	150.0 ± 0.00	**17.27 ± 0.00**	6496.80 ± 346.90	43.31	**1.774 ± 0.089**	2.062 ± 0.291
	CBF	Gurobi + Py	151.0 ± 0.00	17.44 ± 0.00	12146.60 ± 185.49	80.44	0.930 ± 0.014	**-0.018 ± 0.001**
4	BOW	FCL + C++	210.8 ± 6.5	18.67 ± 0.12	**30.90 ± 0.68**	**0.15**	0.945 ± 0.036	0.172 ± 0.028
	DWA	FCL + C++	373.0 ± 0.00	34.33 ± 0.00	169.20 ± 3.58	0.45	0.970 ± 0.00	0.030 ± 0.00
	HRVO	C++	465.0 ± 0.00	21.06 ± 0.00	676.32 ± 12.82	1.45	0.453 ± 0.00	0.341 ± 0.000
	RRT	FCL + C++	**77.9 ± 10.0**	23.68 ± 2.67	745.79 ± 406.64	9.57	0.769 ± 0.021	0.898 ± 0.072
	MPPI	CUDA + Py	198.0 ± 0.00	18.86 ± 0.00	8674.80 ± 156.41	43.81	**1.467 ± 0.026**	0.194 ± 0.010
	CBF	Gurobi + Py	171.0 ± 0.00	**18.09 ± 0.00**	15135.40 ± 335.45	88.51	0.805 ± 0.018	**-0.002 ± 0.000**
5	BOW	FCL + C++	95.0 ± 5.4	**7.56 ± 0.19**	**12.89 ± 0.09**	**0.14**	**1.007 ± 0.077**	0.357 ± 0.052
	DWA	FCL + C++	—	—	—	—	—	—
	HRVO	C++	204.0 ± 0.00	8.34 ± 0.00	369.20 ± 2.49	1.81	0.409 ± 0.000	**0.260 ± 0.00**
	RRT	FCL + C++	**31.7 ± 3.7**	9.53 ± 0.80	93.18 ± 56.97	2.94	0.785 ± 0.035	0.927 ± 0.100
	MPPI	CUDA + Py	139.0 ± 0.00	8.86 ± 0.00	5983.00 ± 183.18	43.04	0.691 ± 0.021	-0.930 ± 0.086
	CBF	Gurobi + Py	—	—	—	—	—	—
6	BOW	FCL + C++	101.9 ± 5.1	**8.73 ± 0.09**	**10.00 ± 0.33**	**0.10**	**0.940 ± 0.063**	0.158 ± 0.029
	DWA	FCL + C++	556.0 ± 0.00	25.36 ± 0.00	589.78 ± 4.04	1.06	0.556 ± 0.00	**0.043 ± 0.00**
	HRVO	C++	—	—	—	—	—	—
	RRT	FCL + C++	**36.1 ± 8.1**	10.54 ± 1.56	128.16 ± 79.40	3.55	0.757 ± 0.041	0.811 ± 0.060
	MPPI	CUDA + Py	1048.0 ± 0.00	29.94 ± 0.00	44312.20 ± 645.95	42.28	0.056 ± 0.001	0.421 ± 0.018
	CBF	Gurobi + Py	—	—	—	—	—	—

TABLE I: Comparison of path planning methods across six environments. Best values for each environment are highlighted in **bold**. A dash (—) indicates that no solution exists for that planner in the specific environment.
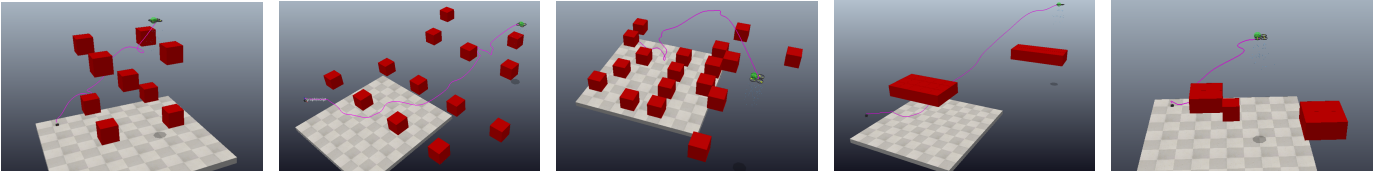

Fig. 5: Simulated 3D trajectory planning scenarios for a UAV using the BOW planner in Coppeliasim Simulator, showing five different environments.
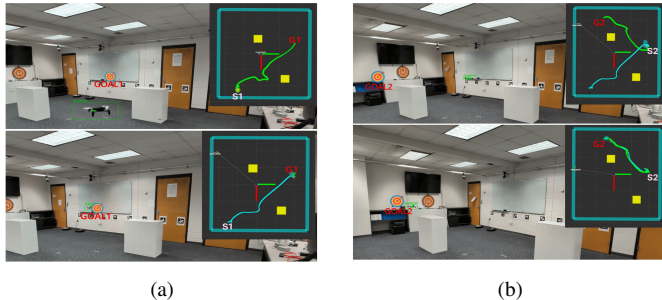

(a)   (b)

Fig. 6: The BOW planner can enhance the autonomous navigation of UAV in real-world scenarios.

Env.	Density	Dispersity	Total Time (ms)	Path Length (m)	Steps
1	0.14	2.38	10.81 ± 0.43	6.73 ± 1.14	132.70 ± 33.95
2	0.04	4.15	12.07 ± 0.31	10.76 ± 0.90	147.90 ± 25.30
3	0.05	3.35	12.84 ± 0.49	10.57 ± 1.01	152.60 ± 30.13
4	0.12	3.35	11.71 ± 0.47	10.77 ± 0.57	151.90 ± 23.53
5	0.09	2.63	5.64 ± 0.18	5.78 ± 1.60	89.50 ± 38.44

TABLE II: Performance Analysis of the BOW Planner for Quadrotor UAV Navigation Across Varied Obstacle Densities and Dispersities.

Table II summarizes the BOW planner's performance for a quadrotor UAV across five environments (Fig. 5) with varying obstacle densities and dispersities. Higher occupied area ratios (e.g., Env. 1: 0.14; Env. 4: 0.12) led to shorter paths but moderate computation and node expansion. In contrast, lower occupied area ratios (e.g., Env. 2: 0.04) resulted in longer paths and greater exploration. High dispersity (e.g., Env. 2: 4.15) increased planning time, indicating greater global search. Env. 5 showed the lowest planning time (5.64 ms), likely due to moderate density (0.09) and dispersity (2.63). Overall, BOW performs best in environments with moderate density and dispersity, balancing path quality and efficiency.

To further validate our methodology, we conducted additional evaluations in complex environments, specifically Bugtrap and Poisson forest settings, as illustrated in Fig. 4.

Complete experimental results for both terrestrial and aerial platforms can be found on our project website.

VI. CONCLUSION

We introduced the BOW planner, a novel contribution to local motion planning in complex, non-convex, and high-dimensional environments through the integration of constrained Bayesian optimization. Grounded in a proven theoretical foundation, the method combines a window-based motion planner with constrained Bayesian optimization to produce near-optimal control policies while minimizing evaluations of computationally expensive constrained objectives. Extensive simulation benchmarks and real-world experiments demonstrate that the BOW planner achieves notable improvements in sample efficiency, trajectory length, and computation time compared to a range of existing local planners. Its deployment across diverse robotic platforms further highlights its online adaptability and safety-aware optimization. The planner is released as open-source software, facilitating adoption and extension by researchers and practitioners engaged in high-dimensional motion planning.

Like most local planners, the BOW planner employs a cost-to-go heuristic to efficiently find feasible trajectories, which may result in getting stuck in local minima when the heuristic is inadmissible (e.g., in narrow passage problems). Future research will focus on combining tree-based search techniques with the BOW planner to address global motion planning problems.

REFERENCES

[1] J. Leu, G. Zhang, L. Sun, and M. Tomizuka, "Efficient robot motion planning via sampling and optimization," in *American Control Conference*, 2021, pp. 4196–4202.

[2] "Receding horizon task and motion planning in changing environments," *Robotics and Autonomous Systems*, vol. 145, p. 103863, 2021.

[3] D. Q. Mayne and H. Michalska, "Model predictive control of nonlinear systems," *1991 American Control Conference*, pp. 2343–2348, 1991.

[4] F. Zhang, N. Li, T. Xue, Y. Zhu, R. Yuan, and Y. Fu, "An improved dynamic window approach integrated global path planning," in *IEEE International Conference on Robotics and Biomimetics*. IEEE, 2019, pp. 2873–2878.

[5] 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.

[6] J. Snape, J. Van Den Berg, S. J. Guy, and D. Manocha, "The hybrid reciprocal velocity obstacle," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 696–706, 2011.

[7] K. Bergman, O. Ljungqvist, T. Glad, and D. Axehill, "An optimization-based receding horizon trajectory planning algorithm," *IFAC-PapersOnLine*, pp. 15 550–15 557, 2020.

[8] T. Nyberg, C. Pek, L. Dal Col, C. Norén, and J. Tumova, "Risk-aware motion planning for autonomous vehicles with safety specifications," in *2021 ieee intelligent vehicles symposium (iv)*. IEEE, 2021, pp. 1016–1023.

[9] G. Costante, C. Forster, J. A. Delmerico, P. Valigi, and D. Scaramuzza, "Perception-aware path planning," *ArXiv*, vol. abs/1605.04151, 2016.

[10] J. Wang, T. Zhang, N. Ma, Z. Li, H. Ma, F. Meng, and M. Q.-H. Meng, "A survey of learning-based robot motion planning," *IET Cyber-Systems and Robotics*, pp. 302–314, 2021.

[11] J. Carvalho, A. T. Le, M. Baierl, D. Koert, and J. Peters, "Motion planning diffusion: Learning and planning of robot motions with diffusion models," in *International Conference on Intelligent Robots and Systems*. IEEE, 2023, pp. 1916–1923.

[12] J. Ichnowski, Y. Avigal, V. Satish, and K. Goldberg, "Deep learning can accelerate grasp-optimized motion planning," *Science Robotics*, p. eabd7710, 2020.

[13] H. Ren and A. H. Qureshi, "Robot active neural sensing and planning in unknown cluttered environments," *IEEE Transactions on Robotics*, pp. 2738–2750, 2023.

[14] L. Li, Y. Miao, A. H. Qureshi, and M. C. Yip, "MPC-MPNet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints," *IEEE Robotics and Automation Letters*, pp. 4496–4503, 2021.

[15] C. Quintero-Pe na, C. Chamzas, V. Unhelkar, and L. E. Kavraki, "Motion planning via bayesian learning in the dark," in *International Conference on Intelligent Robots and Systems*. IEEE, 2021, pp. 5634–5641.

[16] J. Ungredda and J. Branke, "Bayesian optimisation for constrained problems," *ACM Transactions on Modeling and Computer Simulation*, vol. 34, pp. 1–26, 2024.

[17] M. Diessner, J. O'Connor, A. Wynn, S. Laizet, Y. Guan, K. Wilson, and R. D. Whalley, "Investigating bayesian optimization for expensive-to-evaluate black box functions: Application in fluid dynamics," vol. 8, 2022.

[18] M. Ali, H. Jardali, N. Roy, and L. Liu, "Autonomous navigation, mapping and exploration with gaussian processes." in *Robotics: Science and Systems*, 2023.

[19] A. G. Wilson, D. A. Knowles, and Z. Ghahramani, "Gaussian process regression networks," in *International Conference on Machine Learning*, 2012, pp. 1139–1146.

[20] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time gaussian process motion planning via probabilistic inference," *The International Journal of Robotics Research*, vol. 37, no. 11, pp. 1319–1340, 2018.

[21] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious model predictive control using gaussian process regression," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2736–2743, 2019.

[22] O. Andersson, M. Wzorek, P. Rudol, and P. Doherty, "Model-predictive control with stochastic collision avoidance using bayesian policy optimization," in *IEEE International Conference on Robotics and Automation*. IEEE, 2016, pp. 4597–4604.

[23] V. Gabler and D. Wollherr, "Bayesian optimization with unknown constraints in graphical skill models for compliant manipulation tasks using an industrial robot," *Frontiers in Robotics and AI*, vol. 9, 2022.

[24] 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.

[25] A. Schoer, H. Teixeira-Dasilva, C. So, M. Mann, and R. Tron, "Control barrier function toolbox: An extensible framework for provable safety," in *NASA Formal Methods Symposium*. Springer, 2024, pp. 352–358.

[26] M. A. Gelbart, J. Snoek, and R. P. Adams, "Bayesian optimization with unknown constraints," *Conference on Uncertainty in Artificial Intelligence*, pp. 250–259, 2014.

[27] C. A. Micchelli, Y. Xu, and H. Zhang, "Universal kernels," *Journal of Machine Learning Research*, vol. 7, no. 12, 2006.

[28] A. W. van der Vaart and J. H. van Zanten, "Rates of contraction of posterior distributions based on Gaussian process priors," *The Annals of Statistics*, vol. 36, no. 3, pp. 1435 – 1463, 2008.

[29] E. Vazquez and J. Bect, "Convergence properties of the expected improvement algorithm with fixed mean and covariance functions," *Journal of Statistical Planning and Inference*, vol. 140, no. 11, pp. 3088–3095, 2010.

[30] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.