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

***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. Upon acceptance of this manuscript, the source code for the BOW Planner will be made publicly available to facilitate further research and development in the field.**

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, 3]. However, evaluating this constrained objective function over a long planning horizon is computationally expensive, especially for systems with complex dynamics and constraints [2, 4].

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 [5, 6]. Later, a unified motion planning and control solution was introduced, capable of navigating environments with both static and dynamic obstacles [7, 8]. While RHC reduces computational burden and allows for online re-planning, it still faces challenges in cluttered environments where the number of evaluations is limited by available computational resources and the necessity for frequent updates [1, 3, 9].

Recent advancements have explored various techniques to enhance performance within computational constraints, including learning-based receding horizon planning [10], risk-aware planning [11], and perception-aware planning [12]. These methods aim to improve the efficiency and effectiveness of motion planning in complex environments. In this context, Deep learning-based motion planners can be categorized into two types: (i) methods that generate end-to-end collision-free paths or trajectories for the given configuration space, and (ii) methods that improve subsystems of a motion planning framework, including preprocessing, prediction, execution, and collision-checking modules [13]. End-to-end planners like 3D convolution neural networks, PointNet++, and Transformers have been employed in various robot locomotion, visuomotor control, and planning tasks [14, 15, 16]. In contrast, subsystem improvement frameworks like Dynamic MPNets [17, 18] harness the strengths of both neural planning and model predictive control to compute a robust and efficient solution in learned environments. This approach enables the system to generate multiple predictions within a planning window, rather than relying on a single general prediction. However, a common limitation of deep learning-based frameworks is their reliance on prior knowledge of the environment, which can be difficult to obtain. Furthermore, efficiently balancing optimality, safety, and computational efficiency remains an open challenge, particularly in uncertain and dynamic 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 [19]. 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 [20, 21]. Spatial GPs address the curse of dimensionality by selecting a small, spatially diverse subset of the dynamic window, thereby significantly reducing the sample space [22, 23]. They avoid overfitting by considering multiple potential curves for the underlying process. While these techniques perform well for unconstrained objectives, most motion planning problems require faster computation and multiple trajectory options under kinodynamic constraints.

In this paper, we propose BOW planner, a novel constrained Bayesian optimization approach for local motion planning of mobile robots in cluttered environments. The main contributions of this paper are:

1) **Enhanced Sample Efficiency:** BOW can find optimal control policies with fewer evaluations, providing feasible solutions in significantly less time compared to existing methods.
2) **Safety-Integrated Optimization:** Safety constraints are directly incorporated into the optimization process, ensuring solutions are both optimal and feasible.
3) **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.
4) **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 $\mathcal{W} \subset \mathbb{R}^d$ containing N unknown obstacles. The robot's state space is denoted as $\mathcal{X} \subset \mathbb{R}^n$, and its control space is represented by $\mathcal{U} \subset \mathbb{R}^m$.

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 $\mathbf{x} : \{0, \dots, T\} \to \mathcal{X}_{\text{free}}$ 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:

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

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:

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

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 $\mathbf{u}_t^*$ is then applied to the robot, and the process repeats at the next time step.

III. BOW Planner

Given a dynamic planning window $V_d \subset \mathcal{U}$ based on the robot's current velocity and acceleration limits [24], we need to evaluate the objective function J and constraints c_k for each control input $\mathbf{u} = (v, \omega) \in V_d$ by simulating the robot's trajectory over a short time interval $[\tau, \tau + \Delta) \subseteq [0, T]$.

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 compute the cost-to-go function over a short horizon trajectory, we capture the future cost incurred by being in a given state with Δ steps to go:

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

where $J(\mathbf{x}_t, \mathbf{u}_t)$ is the stage cost and $J_F(\mathbf{x}_T)$ is the terminal cost. (5) gives the optimal cost-to-go from any state to the goal. This short horizon trajectory acts as a proxy for the original cost function at the optimization problem (3), reducing the computational cost of computing the entire path. To reduce the computational effort even further, we consider a subset of the possible short-horizon trajectories where the same control is applied i.e., $\mathbf{u}_\tau = \mathbf{u}_t$ for $t = 1, \dots, \Delta$. Then, (5) becomes

$$\mathcal{J}(\mathbf{x}_{[\tau:\tau+\Delta+1]}, \mathbf{u}_{[\tau:\tau+\Delta]}) = \sum_{t=\tau}^{\tau+\Delta} J(\mathbf{x}_t, \mathbf{u}_\tau) + J_F(\mathbf{x}_{\Delta+1}), \tag{6}$$

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

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:

$$c_k(\mathbf{x}_t) = r_{\text{safe}} - d(\mathbf{x}_t, \mathcal{O}_k), \tag{7}$$

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

Let $y = J(\mathbf{x}, \mathbf{u})$, and let $\mathbf{c}(\mathbf{x}) = (c_1(\mathbf{x}), \dots, c_k(\mathbf{x}))$ be the vector of constraint values. For computing the optimal control over the domain V_d, we sample m training data within V_d from the collection of objective function observations, $D_J = \{(\mathbf{u}_i, J(\mathbf{x}_i, \mathbf{u}_i))\}_{i=1}^m$, and constraints, $D_c = \{(\mathbf{u}_i, \mathbf{c}(f(\mathbf{x}_i, \mathbf{u}_i)))\}_{i=1}^m$.

We model the objective function observations as a Gaussian Process (GP) which is fully specified by a mean function $\mu(\mathbf{u}) = \mathbb{E}[y(\mathbf{u}) \mid D_J]$ and its covariance $\text{Cov}[y(\mathbf{u}), y(\mathbf{u}') \mid D_J]$ as:

$$\mathbb{E}[y(\mathbf{u}) \mid D_j] = \mu(\mathbf{u}) = \mu(\mathbf{u}) - k_{y_0}(\mathbf{u}, U) \left(k_{y_0}(U,U) + I\sigma_\epsilon^2\right)^{-1} (Y - \mu(U)), \tag{8}$$

$$\mathrm{Cov}[y(\mathbf{u}), y(\mathbf{u}') \mid D_j] = k_{y_m}(\mathbf{u}, \mathbf{u}') = k_{y_0}(\mathbf{u}, \mathbf{u}') - k_{y_0}(\mathbf{u}, U) \left(k_{y_0}(U,U) + I\sigma_\epsilon^2\right)^{-1} k_{y_0}(U, \mathbf{u}'), \tag{9}$$

where U is the covariance matrix with $U_{ij} = k(\mathbf{u}_i, \mathbf{u}_j)$, and $Y = [y_1, \ldots, y_m]^\top$, The term $I\sigma_\epsilon^2$ is added to the covariance matrix $k_{y_0}(U,U)$ to account for this observation noise.

Similarly, each constraint is modeled as an independent GP on training data $\mathcal{D}_c$ defined by a constraint mean function $\mu(\mathbf{u}) = \mathbb{E}[c_k(f(\mathbf{x},\mathbf{u})) \mid \mathcal{D}_c]$ and covariance $\mathrm{Cov}[c_k(f(\mathbf{x},\mathbf{u})), c_k(f(\mathbf{x},\mathbf{u}')) \mid \mathcal{D}_c]$.

To handle constraints in the optimization process, we use the Constrained Expected Improvement (CEI) acquisition function. CEI balances the expected improvement of the objective function with the probability of satisfying the constraints. The Constrained Expected Improvement for a new point $\mathbf{u}$ is defined as:

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

where $\mathrm{EI}(\mathbf{u})$ is the standard Expected Improvement. $P(\text{feasible} \mid \mathbf{u})$ is the probability that all constraints are satisfied at $\mathbf{u}$. The Expected Improvement (EI) is given by:

$$\mathrm{EI}(\mathbf{u}) = (y^* - \mu(\mathbf{u}))\Phi(Z) + \sigma(\mathbf{u})\phi(Z), \tag{11}$$

where $y^* = \min_{i=1,\ldots,m} y_i$ is the current best observed objective value; $\mu(\mathbf{u})$ is the posterior mean of the GP at $\mathbf{u}$; $\sigma(\mathbf{u}) = \sqrt{k_{y_n}(\mathbf{u},\mathbf{u})}$ is the posterior standard deviation at $\mathbf{u}$; $Z = \frac{y^* - \mu(\mathbf{u})}{\sigma(\mathbf{u})}$. Φ is the cumulative distribution function of the standard normal distribution, and ϕ is the probability density function of the standard normal distribution.

Assuming the constraints are conditionally independent given $\mathbf{u}$, the probability of feasibility is:

$$P(\text{feasible} \mid \mathbf{u}) = \prod_{k=1}^{K} P(c_k(\mathbf{u}) \leq 0 \mid \mathcal{D}_c). \tag{12}$$

For each constraint c_k, modeled as a GP, the probability $P(c_k(\mathbf{u}) \leq 0 \mid \mathcal{D}_c)$ can be computed as:

$$P(c_k(\mathbf{u}) \leq 0 \mid \mathcal{D}_c) = \Phi\left(\frac{-\mu(\mathbf{u})}{\sigma(\mathbf{u})}\right), \tag{13}$$

where $\mu(\mathbf{u})$ and $\sigma(\mathbf{u})$ are the posterior mean and standard deviation of the GP for the k-th constraint at $\mathbf{u}$, respectively.

The optimal control input at time t to evaluate is chosen by maximizing the CEI:

$$\mathbf{u}_t^* = \arg\max_{\mathbf{u} \in V_d} \mathrm{CEI}(\mathbf{u}). \tag{14}$$

Equation (14) ensures that the optimization process not only seeks to improve the objective function but also respects the constraints by considering the probability of feasibility.

Algorithm 1 BOW Planning and Control

Input: Initial state $\mathbf{x}_0$, goal $\mathbf{x}_g$, collision checker cc, parameters pm
Output: Solution found, final trajectory
Initialize $t \leftarrow 0$, $\mathbf{x}_t \leftarrow \mathbf{x}_0$, final_result $\leftarrow \emptyset$
while $\|\mathbf{x}_t - \mathbf{x}_g\| \leq$ goal_radius **do**
  Define planning window $V_d \subset \mathcal{U}$ using robot dynamics and parameters pm
  Sample m controls $\{\mathbf{u}_i\}_{i=1}^m \in V_d$ using CBO
  Initialize datasets: $D_j \leftarrow \emptyset$ and $D_c \leftarrow \emptyset$
  for each $\mathbf{u}_i$ **do**
    Simulate $\mathbf{x}_{t+1}^{(i)} = f(\mathbf{x}_t, \mathbf{u}_i)$ using Eqn. (1)
    Evaluate stage cost $y_i = J(\mathbf{x}_t, \mathbf{u}_i) = \|\mathbf{x}_g - \mathbf{x}_{t+1}^{(i)}\|$
    Compute constraint values $c_k^{(i)} = r_{\text{safe}} - d(\mathbf{x}_{t+1}^{(i)}, O_k)$ for all k
    $D_J \leftarrow D_J \cup \{\mathbf{u}_i, y_i\}$ and $D_c \leftarrow D_c \cup \{\mathbf{u}_i, \mathbf{c}^{(i)}\}$
  end for
  Train GPs on D_J and D_c
  Compute CEI using Eqns. (10), (13), (14):
  Choose $\mathbf{u}_t^* = \arg\max_{\mathbf{u} \in V_d} \mathrm{CEI}(\mathbf{u})$
  Generate trajectory traj $= \{\mathbf{x}_t, \ldots, \mathbf{x}_{t+T}\}$ using RK4 and $\mathbf{u}_t^*$
  if traj is collision-free and non-empty **then**
    Set $\ell = \min(\text{len(traj)} - 1, \text{pref\_speed\_index})$
    Update $\mathbf{x}_t \leftarrow \text{traj}[\ell]$
    Append traj$[0:\ell]$ to final_result
  end if
  $t \leftarrow t + 1$
end while
return true, final_result

D. 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.

At each iteration, a BOW Planner is instantiated with the current state $\mathbf{x}_t$, the goal, a collision checker, and a set of parameters. This planner leverages Bayesian Optimization, utilizing a GP and the CEI acquisition function to compute the optimal control $\mathbf{u}_t^*$. The CEI function enables effective constraint handling by modeling both the objective function and the constraints through the GP, ensuring that the selected control adheres to feasibility requirements such as avoiding obstacles. Using the computed optimal control $\mathbf{u}_t^*$, a receding horizon trajectory is generated via the motion model, implemented through RK4 integration. This trajectory represents a sequence of predicted states over the planning horizon. The algorithm then checks if this trajectory is both collision-free (as verified by the collision checker) and non-empty. If these conditions are met, the algorithm determines an index ℓ. This index ℓ allows the approach to select a control action within the planning horizon that aligns closely with the

robot's preferred velocity, offering a more flexible strategy compared to conventional Model Predictive Control (MPC), which typically applies only the immediate control action. The current state $\mathbf{x}_t$ is then updated to the state at index ℓ in the trajectory, and the segment of the trajectory up to ℓ index is appended to the final trajectory. This iterative update ensures that only feasible control actions —those that maintain a collision-free path, are executed, minimizing the risk of collisions with obstacles. Unlike traditional methods that might reject a control and re-optimize if constraints are violated, this approach inherently generates feasible trajectories by integrating constraint satisfaction into the planning process via the CEI function.

The loop continues by evaluating the distance between the updated state $\mathbf{x}_t$ and the goal. If this distance is less than or equal to the goal radius, the algorithm terminates, indicating that the robot has successfully reached its target. Upon breaking the loop, the algorithm returns a success flag and the complete final trajectory, which encapsulates the robot's path from the initial state to the goal. This iterative methodology, powered by Bayesian Optimization and the CEI acquisition function, ensures robust constraint management within the receding horizon framework.

IV. Algorithm Analysis

Lemma 1. *Let $f : V_d \to \mathbb{R}$ be a Lipschitz continuous function on a compact set V_d. If a GP with a squared exponential kernel models f, and observations are corrupted by bounded noise $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$, then the posterior mean $\mu_m(\mathbf{u})$ converges uniformly to $f(\mathbf{u})$ as $m \to \infty$.*

Sketch of Proof. The squared exponential kernel ensures smoothness consistent with Lipschitz continuity. As $m \to \infty$ with dense sampling in V_d, the posterior variance $\sigma_m^2(\mathbf{u}) \to 0$ uniformly due to the kernel's properties and bounded noise. The posterior mean $\mu_m(\mathbf{u})$ approximates $f(\mathbf{u})$ with increasing accuracy, converging uniformly by GP approximation properties. □

Theorem 1. *Let $V_d \subset \mathcal{U}$ be a compact dynamic planning window defined by the robot's velocity and acceleration limits. Suppose the objective function $J(\mathbf{u})$ and constraint functions $c_k(\mathbf{u})$ for $k = 1, \ldots, K$ are Lipschitz continuous over $\mathbf{u} \in V_d$. Assume the system dynamics f are Lipschitz continuous, and the Gaussian Process (GP) models for J and c_k use kernels (e.g., squared exponential) with bounded observation noise. Then, as the number of samples $m \to \infty$, the control input $\mathbf{u}_t^*$ selected by maximizing the Constrained Expected Improvement (CEI) converges to the true optimal control input $\mathbf{u}_{opt}$ that minimizes $J(\mathbf{u})$ within V_d while satisfying all constraints $c_k(\mathbf{u}) \leq 0$.*

Proof. We prove this theorem in five steps, leveraging properties of Gaussian Processes and Bayesian optimization under the given assumptions.

Step 1: GP Approximation and Uniform Convergence: Since $J(\mathbf{u})$ and $c_k(\mathbf{u})$ are Lipschitz continuous on the compact set V_d, their GP models converge to the true functions as the number of samples increases. Specifically, the posterior mean $\mu_{J,m}(\mathbf{u}) \to J(\mathbf{u})$ and $\mu_{c_k,m}(\mathbf{u}) \to c_k(\mathbf{u})$ uniformly as $m \to \infty$. Additionally, the posterior variances $\sigma_{J,m}^2(\mathbf{u}) \to 0$ and $\sigma_{c_k,m}^2(\mathbf{u}) \to 0$ uniformly over V_d. This follows from the universal approximation capability of GPs with appropriate kernels (e.g., squared exponential) and dense sampling in V_d.

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

$$P(c_k(\mathbf{u}) \leq 0 \mid \mathcal{D}_c) = \Phi\left(\frac{-\mu_{c_k,m}(\mathbf{u})}{\sigma_{c_k,m}(\mathbf{u})}\right), \tag{15}$$

where Φ is the standard normal cumulative distribution function, and $\mathcal{D}_c$ is the dataset of constraint observations. As $m \to \infty$, $\mu_{c_k,m}(\mathbf{u}) \to c_k(\mathbf{u})$ and $\sigma_{c_k,m}(\mathbf{u}) \to 0$. Thus,

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

The total probability of feasibility is given by

$$P(\text{feasible} \mid \mathbf{u}) = \prod_{k=1}^{K} P(c_k(\mathbf{u}) \leq 0 \mid \mathcal{D}_c). \tag{16}$$

In the feasible region where $c_k(\mathbf{u}) \leq 0$ for all k (assuming strict feasibility $c_k(\mathbf{u}) < 0$ for simplicity), $P(\text{feasible} \mid \mathbf{u}) \to 1$. 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 Expected Improvement (EI): The Expected Improvement for the objective $J(\mathbf{u})$ is defined as

$$\text{EI}(\mathbf{u}) = (y^* - \mu_{J,m}(\mathbf{u}))\Phi(Z) + \sigma_{J,m}(\mathbf{u})\phi(Z), \tag{17}$$

where $Z = \frac{y^* - \mu_{J,m}(\mathbf{u})}{\sigma_{J,m}(\mathbf{u})}$, $y^* = \min_{\mathbf{u}' \in \{\mathbf{u}_1, \ldots, \mathbf{u}_m\}} J(\mathbf{u}')$ is the current best observed value, and ϕ is the standard normal probability density function. As $m \to \infty$, $\mu_{J,m}(\mathbf{u}) \to J(\mathbf{u})$ and $\sigma_{J,m}(\mathbf{u}) \to 0$. Furthermore, $y^* \to \min_{\mathbf{u} \in V_d} J(\mathbf{u})$ due to dense sampling in the compact set V_d. Thus, $\text{EI}(\mathbf{u}) \to \max(y^* - J(\mathbf{u}), 0)$, which is positive only when $J(\mathbf{u}) < y^*$, guiding the search toward better points.

Step 4: Behavior of Constrained Expected Improvement (CEI): The CEI combines EI with feasibility:

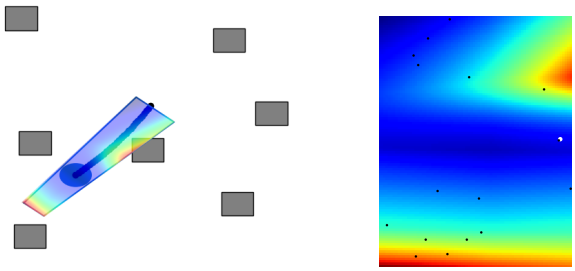
$$\text{CEI}(\mathbf{u}) = \text{EI}(\mathbf{u}) \cdot P(\text{feasible} \mid \mathbf{u}). \tag{18}$$

As $m \to \infty$, for feasible $\mathbf{u}$, $P(\text{feasible} \mid \mathbf{u}) \to 1$, so $\text{CEI}(\mathbf{u}) \to \text{EI}(\mathbf{u})$, focusing on improving $J(\mathbf{u})$. For infeasible $\mathbf{u}$, $P(\text{feasible} \mid \mathbf{u}) \to 0$, so $\text{CEI}(\mathbf{u}) \to 0$, effectively excluding these points.

Step 5: Convergence to the Optimal Solution: The selected control input is

$$\mathbf{u}_t^* = \arg\max_{\mathbf{u} \in V_d} \text{CEI}(\mathbf{u}). \tag{19}$$

As $m \to \infty$, the GP models become exact ($\mu_{J,m} \to J$, $\mu_{c_k,m} \to c_k$), and CEI identifies feasible points with the greatest potential to minimize J. Since V_d is compact and J is continuous, the constrained optimum $\mathbf{u}_{\text{opt}} = \arg\min_{\mathbf{u} \in V_d,\, c_k(\mathbf{u}) \leq 0} J(\mathbf{u})$ exists. CEI-based optimization converges to $\mathbf{u}_{\text{opt}}$ under the given conditions. Thus, $\mathbf{u}_t^* \to \mathbf{u}_{\text{opt}}$ as $m \to \infty$, completing the proof. □



(a) (b)

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 fewer samples (indicated by black dots).

A. Computational Complexity of BOW Planner

The BOW Planning algorithm iteratively computes an optimal control $\mathbf{u}_t^*$ using Bayesian Optimization with Gaussian Processes (GP) and generates collision-free trajectories over Ψ iterations. With a single optimization iteration ($\psi = 1$) and a constant number of GP samples m, the complexity per iteration includes:

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

Total complexity is:

$$O(\Psi \cdot (m^3 + H \cdot (n + \log C))) = O(\Psi \cdot H \cdot (n + \log C))$$

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

V. Experiments and Results

The experiments were conducted on a desktop computer equipped with AMD Ryzen™ 9 7950X Desktop Processor and an NVIDIA GTX 4070 GPU, along with 32 GB of RAM. The system ran on Ubuntu 22.04 LTS. All robots are localized with a VICON Motion Capture System with 120 Hz frequency.

A. Sample Efficiency

The sample efficiency property of the BOW planner is further illustrated in Fig. 1. Given a dynamic window, 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. Figure 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 robot navigating through a 2-D space containing obstacles. We performed simulations for ground vehicles across five different environments, as shown in Figure 2. These same environments were also replicated in the CoppeliaSim simulator. The robot'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.

Fig. 3a illustrates the BOW planner's implementation on a Roomba robot navigating an environment with obstacles marked in red. The robot's goal location is indicated by a cyan "**G**", and the figure shows the robot's path as it avoids obstacles to reach its destination. Due to the faster approximation of the optimal control, we can scale the BOW planner for distributed robot navigation problems. Fig. 3b presents a scenario with two Roomba robots. The gray Roomba aims for a gray "**G**" while the cyan Roomba targets a cyan "**G**". As in Fig. 3a, both robots navigate around obstacles to reach their goals. The figures demonstrate the robots' obstacle avoidance capabilities and their navigation to specific destinations. The distinct colors for the robots and their goals in Fig. 3b helps differentiate their paths. In this scenario, each Roomba treats the other as a dynamic obstacle, adjusting its path to avoid collisions while moving toward its goal.

C. Benchmark

We evaluate the performance of the BOW planner against five established path planning algorithms: Rapidly-exploring Random Tree (RRT) [25], Dynamic Window Approach (DWA) [24], Model Predictive Path Integral (MPPI) [26], Hybrid Reciprocal Velocity Obstacles (HRVO) [27], and Control Barrier Functions (CBF) [28]. Our benchmark assessment compares the BOW planner's efficacy across environments with varying obstacle configurations through key metrics including solution steps, computation time, trajectory length, and per-step execution efficiency. Simulations were conducted in a 20 m $\times$ 20 m workspace, with a robot modeled as having a 0.345 m radius. Environmental complexity was quantified using two metrics: (i) **Obstacle Density**, defined as the ratio of total obstacle area to workspace area, ranging from 5.02% to 8.30%, and (ii) **Obstacle Dispersity**, measured as the variance of local obstacle densities (2.59% to 6.20%), with higher values indicating greater clustering, as shown in Table I.

For each environment and planner combination, we conducted 10 independent runs and calculated the mean and standard deviation of all key metrics. Our comprehensive evaluation across five distinct environments reveals that the

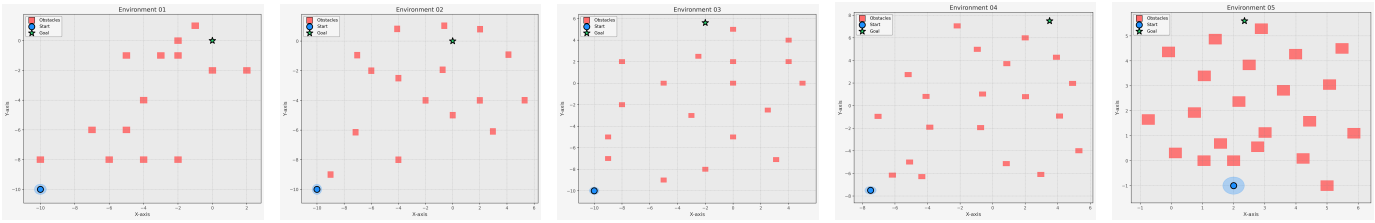


Fig. 2: This figure displays five different environments, with Environment 1 on the left and Environment 5 on the right, arranged in chronological order. The blue sphere represents the start location, the green star indicates the goal location, and the red blocks are obstacles.

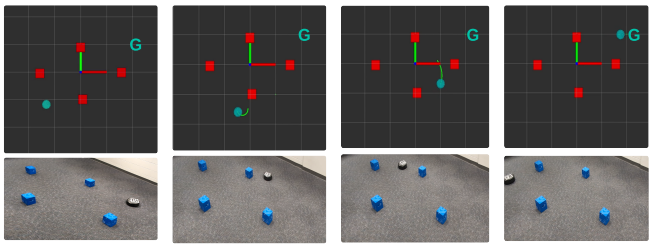
Env	Method	Lib. + Lang	Steps	Traj. Length (m)	Total Time (ms)	Time / Step (ms)	Density (%)	Dispersity (%)
1	BOW	FCL + C++	203.6 ± 24.87	14.81 ± 0.56	**55.6** ± 39.0	**0.27**	5.95	4.39
	HRVO	C++	539.0 ± 4.23	15.78 ± 0.00	260.0 ± 2.96	0.48		
	RRT	FCL + C++	342.0 ± 64.89	17.79 ± 2.56	584.0 ± 321.0	1.71		
	DWA	FCL + C++	157.0 ± 1.23	14.30 ± 0.00	135.0 ± 2.31	0.86		
	MPPI	CUDA + Py	**146.00**	**13.885** ± 0.0	9705.40 ± 131.91	66.48		
	CBF	Gurobi + Py	155.0	14.42 ± 0.0	8837.7 ± 124.7	57.02		
2	BOW	FCL + C++	184.4 ± 8.55	14.56 ± 0.13	**35.9** ± 3.26	**0.19**	5.60	5.16
	HRVO	C++	341.0 ± 2.68	14.87 ± 0.00	286.0 ± 5.77	0.84		
	RRT	FCL + C++	358.1 ± 33.60	19.38 ± 2.56	816.0 ± 280.0	2.28		
	DWA	FCL + C++	237.0 ± 1.86	20.83 ± 0.00	74.3 ± 3.25	0.31		
	MPPI	CUDA + Py	**149.00**	**13.96** ± 0.0	9977.90 ± 129.77	66.97		
	CBF	Gurobi + Py	—	—	—	—		
3	BOW	FCL + C++	168.4 ± 12.66	13.50 ± 0.39	**28.2** ± 1.25	**0.17**	5.02	6.20
	HRVO	C++	260.0 ± 2.04	**12.95** ± 0.00	511.0 ± 4.61	1.96		
	RRT	FCL + C++	292.2 ± 28.50	15.85 ± 1.03	644.0 ± 190.0	2.20		
	DWA	FCL + C++	185.0 ± 1.45	15.66 ± 0.31	112.0 ± 0.23	0.61		
	MPPI	CUDA + Py	**128.00**	13.03 ± 0.0	8583.6 ± 109.14	67.06		
	CBF	Gurobi + Py	175.0	13.482 ± 0.0	12419.3 ± 141.91	70.97		
4	BOW	FCL + C++	260.2 ± 18.28	19.49 ± 0.72	**48.2** ± 2.84	**0.19**	5.95	5.65
	HRVO	C++	465.0 ± 3.65	21.06 ± 0.00	682.0 ± 13.9	1.47		
	RRT	FCL + C++	491.8 ± 103.32	26.76 ± 4.59	1153.0 ± 1183.0	2.34		
	DWA	FCL + C++	374.0 ± 2.94	34.43 ± 0.00	209.0 ± 0.36	0.56		
	MPPI	CUDA + Py	204.00	18.91 ± 0.0	10863.30 ± 185.08	53.25		
	CBF	Gurobi + Py	**183.0**	**18.68** ± 0.0	11636.7 ± 143.64	63.59		
5	BOW	FCL + C++	299.5 ± 88.33	13.99 ± 4.22	**78.6** ± 16.3	**0.26**	8.30	2.59
	HRVO	C++	**164.0** ± 1.29	**6.88** ± 0.00	318.0 ± 6.08	1.94		
	RRT	FCL + C++	199.2 ± 32.97	10.81 ± 1.30	202.0 ± 66.5	1.01		
	DWA	FCL + C++	—	—	—	—		
	MPPI	CUDA + Py	486.00	9.91 ± 0.0	25687.50 ± 490.50	52.85		
	CBF	Gurobi + Py	—	—	—	—		

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

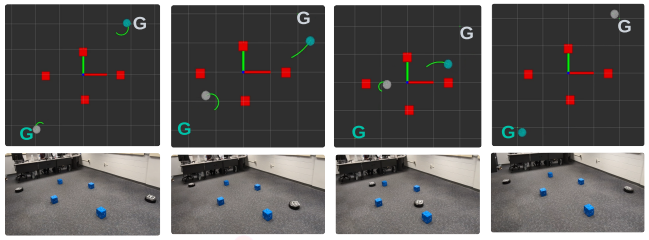
BOW planner (implemented with FCL + C++) consistently demonstrated superior computational efficiency, achieving the fastest total planning times (28.2 ± 1.25 to 78.6 ± 16.3 ms) and per-step execution times (0.17 to 0.27 ms) while maintaining competitive trajectory lengths (13.50 ± 0.39 to 19.49 ± 0.72 m). In contrast, while MPPI frequently produced the shortest paths in simpler environments (Environments 1-3) and CBF excelled in more complex scenarios (Environment 4), both methods showed substantially higher computational demands (CBF: 8837.7 ± 124.7 to 12419.3 ± 141.91 ms total, 57.02 to 70.97 ms per step; MPPI: 8583.6 ± 109.14 to 25687.50 ± 490.50 ms total, 53.25 to 67.06 ms per step). This highlights significant efficiency trade-offs when selecting planning methods. The HRVO algorithm demonstrated competitive performance in Environment 5 with the shortest trajectory (6.88 ± 0.00 m), though it generally required more planning steps in other environments. RRT consistently produced longer trajectories than other methods while requiring moderate computational resources. DWA showed variable performance across environments and failed to generate solutions in Environment 5. Notably, CBF was unable to find solutions in Environments 2 and 5. These results demonstrate that while optimization-based approaches like CBF and MPPI can produce high-quality solutions in certain scenarios, BOW dominates in computational efficiency, and specific planners had completeness issues in particular environments. BOW's consistent performance across all test environments makes it particularly suitable for resource-constrained real-time applications requiring reliable operation across diverse environmental conditions.

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. 4). 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



(a)



(b)

Fig. 3: The BOW planner is applied to UGVs for navigating environments with unknown static obstacles 3a. The planner also has ability to avoid dynamic obstacles, leveraging its rapid planning capabilities 3b.

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.

Figure 4 depicts four 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. Figure 5 illustrates the implementation of the BOW planner in a UAV's navigation through two distinct environments. 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.

Figure 5a and Figure 5b 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.	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 presents the performance of the BOW planner for a quadrotor UAV which was evaluated across five different environments with varying obstacle densities and dispersities. Four of them are shown in Figure.4. The results indicate that higher obstacle densities (e.g., Env 1, 0.14 and Env 4, 0.12) generally lead to shorter path lengths but require moderate computational time and node expansions. Conversely, environments with lower densities, such as Env 2 (0.04), resulted in longer path lengths and increased node exploration. Obstacle dispersity also plays a significant role, with higher dispersity (e.g., Env 2, 4.15) leading to longer paths and increased planning time, suggesting greater global exploration. Notably, Env 5 demonstrated the lowest computational time (5.64 ms), likely due to its moderate density (0.09) and dispersity (2.63), allowing for efficient navigation. Overall, the BOW planner performs optimally in moderate-density, moderate-dispersity environments, balancing path efficiency and computational cost.

VI. Conclusion

We proposed the BOW planner, a computationally efficient approach for achieving near-optimal solutions in high-dimensional constrained motion planning problems. By integrating a window-based motion planner with constrained Bayesian optimization, the BOW planner efficiently achieves optimal control policies with minimal evaluations of an expensive constrained objective function. It provides an objective function, offering theoretical guarantees of near-optimal solutions. Extensive benchmarking shows BOW planner's significant improvements in sample size, trajectory lengths, and solution times compared to various local planners. Implemented on diverse robotic systems, the BOW exhibits online planning capabilities with sample efficiency, safety-aware optimization, and robustness to uncertainty. The BOW planner, like other sampling-based motion planners, faces challenges in ensuring strong safety and optimality guarantees with limited samples. Future research will focus on addressing this limitation to improve the planner's reliability and expand its application to underwater vehicles.

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] N. Castaman, E. Pagello, E. Menegatti, and A. Pretto, "Receding horizon task and motion planning in changing environments," *Robotics and Autonomous Systems*, vol. 145, p. 103863, 2021.

[3] 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.

[4] M. Mansouri, F. Pecora, and P. Schüller, "Combining task and motion planning: Challenges and guidelines," *Frontiers in Robotics and AI*, vol. 8, 05 2021.

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

[6] T. Horton, M. Morales, L. Tapia, R. Pearce, and N. M. Amato, "Feature-sensitive motion planning," 2005.

[7] O. Andersson, O. Ljungqvist, M. Tiger, D. Axehill, and F. Heintz, "Receding-horizon lattice-based motion planning with dynamic obstacle avoidance," in *2018 IEEE*

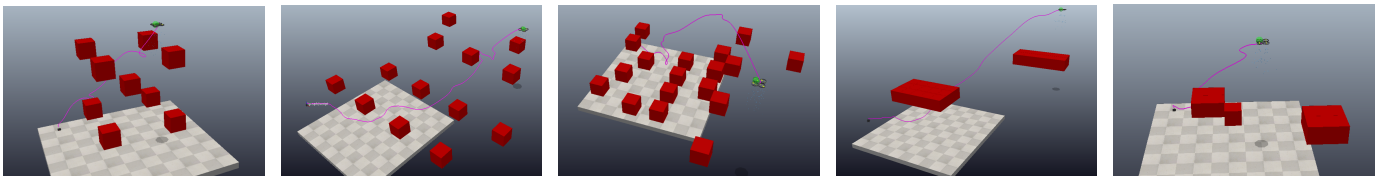
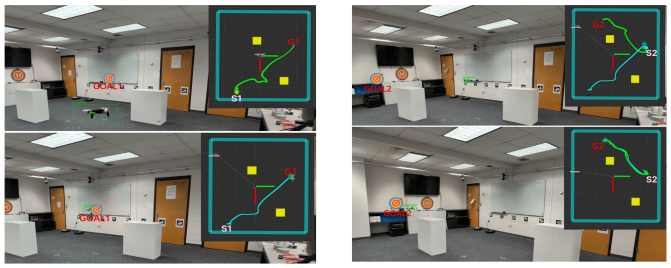


Fig. 4: Simulated 3D trajectory planning scenarios for a UAV using the BOW planner in Coppeliasim Simulator, showing five different environments.



(a) (b)

Fig. 5: The BOW planner can enhance the autonomous navigation of UAV in real-world scenarios.

Conference on Decision and Control (CDC), 2018, pp. 4467–4474.

[8] 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.

[9] F. Berkenkamp, A. Krause, and A. P. Schoellig, "Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics," *Machine Learning*, vol. 112, pp. 3713 – 3747, 2016.

[10] K. Bergman, O. Ljungqvist, T. Glad, and D. Axehill, "An optimization-based receding horizon trajectory planning algorithm," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15 550–15 557, 2020.

[11] T. Nyberg, C. Pek, L. Dal Col, C. Norén, and J. Tumova, "Risk-aware motion planning for autonomous vehicles with safety specifications," in *IEEE Intelligent Vehicles Symposium*, 2021, pp. 1016–1023.

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

[13] 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*, vol. 3, no. 4, pp. 302–314, 2021.

[14] 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.

[15] J. Ichnowski, Y. Avigal, V. Satish, and K. Goldberg, "Deep learning can accelerate grasp-optimized motion planning," *Science Robotics*, vol. 5, no. 48, p. eabd7710, 2020.

[16] H. Ren and A. H. Qureshi, "Robot active neural sensing and planning in unknown cluttered environments," *IEEE Transactions on Robotics*, vol. 39, no. 4, pp. 2738–2750, 2023.

[17] 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*, vol. 6, no. 3, pp. 4496–4503, 2021.

[18] Z. Jian, S. Zhang, L. Sun, W. Zhan, N. Zheng, and M. Tomizuka, "Long-term dynamic window approach for kinodynamic local planning in static and crowd environments," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3294–3301, 2023.

[19] 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.

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

[21] 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," in *Frontiers in Applied Mathematics and Statistics*, 2022.

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

[23] A. G. Wilson, D. A. Knowles, and Z. Ghahramani, "Gaussian process regression networks," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, 2012, pp. 1139–1146.

[24] 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.

[25] P. Cheng, Z. Shen, and S. M. LaValle, "Rrt-based trajectory design for autonomous automobiles and spacecraft," *Archives of control science*, vol. 11, no. 3/4, pp. 167–194, 2001.

[26] 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.

[27] 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.

[28] 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.

TABLE III: Notation Table

Notation	Description
RHC	Receding Horizon Controllers
BOW Planner	Bayesian Optimization over Windows Planner
$W \subset \mathbb{R}^d$	d-dimensional environment
$\mathcal{X} \subset \mathbb{R}^n$	Robot's state space
$\mathcal{U} \subset \mathbb{R}^m$	Robot's control space
$x_t \in \mathcal{X}$	Robot's current state at time t
$u_t \in \mathcal{U}$	Control input
x_{t+1}	Robot's next state
$f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$	State transition function
$x : \{0, ..., T\} \to \mathcal{X}_{free}$	Collision-free trajectory
$x_I \in \mathcal{X}$	Initial state
$x_g \in \mathcal{X}_{goal} \subset \mathcal{X}$	Desired goal state
$\mathcal{X}_{free} \subseteq \mathcal{X}$	Obstacle-free region of the state space
T	Trajectory duration
$J : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$	Cost-to-go function
$c_k : \mathcal{X} \to \mathbb{R}$	Constraints
K	Number of constraints
B	Time budget to compute control inputs
u_t^*	Recommended control input
$V_d \subset \mathcal{U}$	Dynamic planning window
$u = (v, w) \in V_d$	Control input
$x_{t+1} = f(x_t, u_t)$	Predicted state
$J(x_t, u_t)$	Stage cost
$J_f(x_T)$	Terminal cost
$\mathcal{O}_k$	k-th obstacle in the environment
$d(x_t, \mathcal{O}_k)$	Distance between the robot's state x_t and the obstacle $\mathcal{O}_k$
r_{safe}	Safety radius
y	Objective function
c	Vector of constraint values
$D_J = \{(u_i, y_i)\}_{i=1}^m$	Objective function observations
$D_c = \{(u_i, c(f(x_i, u_i)))\}_{i=1}^m$	Constraints
GP	Gaussian Process
$\mu(u) = \mathbb{E}[y(u)\|D_J]$	Mean function
$Cov[y(u), y(u')\|D_J]$	Covariance
U	Covariance matrix with $U_{ij} = k(u_i, u_j)$
$Y = [y_1, ..., y_m]^\top$	
$I\sigma_\epsilon^2$	Observation noise
$\mu(u) = \mathbb{E}[c_k(f(x, u))\|D_c]$	Constraint mean function
$Cov[c_k(f(x, u)), c_k(f(x, u')\|\mathcal{D}_c]$	Constraint covariance
CEI(u)	Constrained Expected Improvement
EI(u)	Expected Improvement
$y^* = min_{i=1,...,m} y_i$	Current best observed objective value
$\mu(u)$	Posterior mean of the GP at u
$\sigma(u) = \sqrt{k_{y_n}(u, u)}$	Posterior standard deviation at u
$Z = \frac{y^* - \mu(u)}{\sigma(u)}$	
$\Phi(\cdot)$	Cumulative distribution function of the standard normal distribution
$\phi(\cdot)$	Probability density function of the standard normal distribution
$P(feasible\|u)$	Probability that all constraints are satisfied at u
$P(c_k(u) \leq 0\|\mathcal{D}_c)$	
$\mu(u)$	Posterior mean
$\sigma(u)$	Posterior standard deviation
u_t^*	Optimal control
traj	Receding horizon trajectory
l	Index
$f : V_d \to \mathbb{R}$	Lipschitz continuous function
V_d	Compact set
$\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$	Bounded noise
$\mu_m(u)$	Posterior mean
$\sigma_m^2(u)$	Posterior variance
$J(u)$	Objective function
$c_k(u)$	Constraint functions
f	System dynamics
u_{opt}	True optimal control input
$\mu_{J,m}(u)$	Posterior mean
$\mu_{c_k,m}(u)$	Posterior mean
$\sigma_{J,m}^2(u)$	Posterior variances
$\sigma_{c_k,m}(u)$	Posterior variances
$\mathcal{D}_c$	Dataset of constraint observations
$y^* = min_{u' \in \{u_1,...,u_m\}} J(u')$	Current best observed value
H	Horizon
n	State dimension
C	Number of geometric primitives in the hierarchy
T	Number of iterations