GPU-Enabled Parallel Trajectory Optimization Framework for Safe Motion Planning of Autonomous Vehicles

Yeongseok Lee, Keun Ha Choi, and Kyung-Soo Kim

***Abstract*—This letter presents a GPU-enabled parallel trajectory optimization framework for model predictive control (MPC) in complex urban environments. It fuses the advantages of sampling-based MPC, which can cope with nonconvex costmaps through random sampling of trajectories, with the advantages of gradient-based MPC, which can generate smooth trajectories. In addition, we leverage a generalized safety-embedded MPC problem definition with a discrete barrier state (DBaS). The proposed framework has three steps: 1) a costmap builder to generate the barrier function map, 2) a seed trajectory generator to choose randomly generated trajectories to send to the optimizers, and 3) a batch trajectory optimizer to optimize each of the seed trajectories and select the best trajectory. Experiments with real-time simulations compare the effectiveness of the proposed framework, sampling-based MPC, and gradient-based MPC, which optimizes a single trajectory. The experiments also compare the application of two different control sequence sampling schemes to the proposed framework. The results show that the proposed framework performs gradient-based optimization but can plan a better trajectory even in complex environments by providing various initial guesses. We also show that the proposed framework can perform more accurate control actions than sampling-based MPC.**

***Index Terms*—Autonomous vehicle navigation, motion planning, model predictive control, hardware acceleradtion.**

I. Introduction

METHODS using trajectory optimization for safe local motion planning of autonomous vehicles in a few seconds, also known as model predictive control (MPC) methods, are widely used [1], [2]. The advantage of MPC methods is that they allow intuitive definitions, such as the mathematical model of the system, hard or soft constraints on objects and its uncertainties [3], to be applied to the optimization framework. However, most of these methods assume a certain level of object detection, which is challenging in complex urban environments. First, it becomes difficult to construct a cost or constraint for avoidance when the detection capability of the method is weakened, such as when the drivable area is complex, when parts of the object are occluded, or when there are objects on which the method has not been trained. Second, determining an appropriate local motion plan and control input from the complex shape of the nonconvex costmap is difficult, and doing so in real time is even more challenging.

MPC strategies can be broadly classified into gradient-based and sampling-based strategies according to the optimization method. Gradient-based MPC [4], [5] is a method that obtains a collision avoidance trajectory through optimization techniques such as sequential quadratic programming [6], and differential dynamic programming [7]. The advantage of this method is that it is suitable for obtaining high-quality trajectories with accurate control actions that are possible once optimized; however, the disadvantage is that if the cost or constraint is nonconvex, escaping from local minima and real-time solving are difficult [1]. There are methods to overcome the nonconvextiy of the problem, such as separately finding convex hulls in the costmap [8] and performing convex region partitioning directly from LIDAR sensor data [9]; however, these methods are difficult to generalize in complex environments or are difficult to run in real time.

In this regard, sampling-based MPC, represented by the model predictive path integral (MPPI) [10], is gaining attention. MPPI is more flexible in dealing with nonconvexity because it does not need to find the gradient during optimization. Additionally, because it generates random trajectory samples, it naturally explores various possibilities for finding the optimal path. However, when applied to real systems, the accuracy of the control action is reduced with chatter [11], sometimes resulting in infeasible trajectories [12]. It is also not free from local minima issues. To solve these problems, measures such as smoothing controls [11], adding an auxiliary controller [13], or modifying the sampling formula [14] are needed.

From this perspective, we aim to combine the strengths of MPPI's random trajectory sampling with those of gradient-based optimization methods in scenarios with nonconvex costmaps. The multistart global optimization method [15], [16] is noteworthy in this regard, which initiates optimization from variety of random initial guesses [17], [18] and find best results from multiple local minima found. As GPU parallel processing power has increased, there have been efforts to process these sequences in parallel to increase real-time computational performance [19].

In this letter, inspired by the multistart method, we propose an MPC framework based on parallel processing for effective trajectory synthesis in complex urban environments. The proposed

Received 7 May 2024; accepted 23 September 2024. Date of publication 30 September 2024; date of current version 11 October 2024. This article was recommended for publication by Associate Editor Y. Jia and Editor A. Banerjee upon evaluation of the reviewers' comments. *(Corresponding author: Kyung-Soo Kim.)*

The authors are with the Department of Mechanical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, Republic of Korea (e-mail: yeongseok94@kaist.ac.kr; choiha99@kaist.ac.kr; kyungsookim@kaist.ac.kr).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2024.3471452, provided by the authors.

Digital Object Identifier 10.1109/LRA.2024.3471452

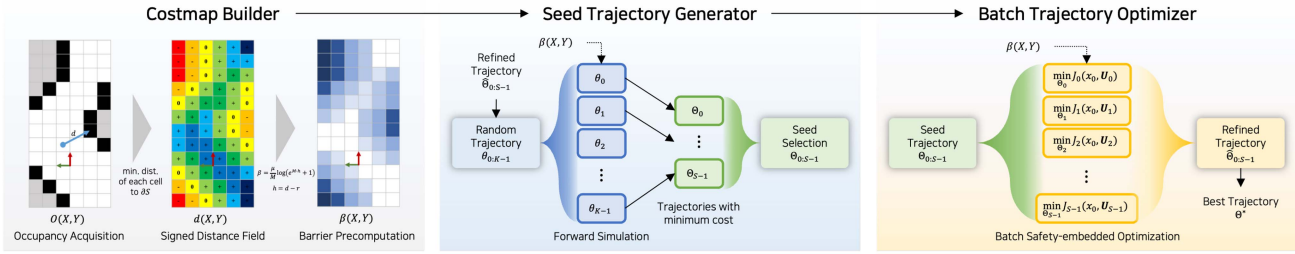


Fig. 1. The overall process of the proposed parallel trajectory optimization framework. First, the costmap builder finds the barrier function $\beta(X,Y)$ in grid notation. Second, the seed trajectory generator selects $\Theta_{0:S-1}$, the trajectories to be used in the optimization, from a set of random trajectory samples $\theta_{0:K-1}$. Finally, the lowest-cost trajectory is selected from $\hat{\Theta}_{0:S-1}$, which are the results after the selected trajectories have passed through the solver.

control framework is characterized by sampling a large number of trajectories and using some of them with least trajectory costs as seeds for a batch gradient-based MPC solver. The motion planning framework we propose consists of three steps, as follows.

1) *Costmap Builder:* Generate a cost map for optimizers based on grid notation [20] suitable for generalized optimization problems that is usable in complex environments where the detection capability is limited.
2) *Seed Trajectory Generator:* Generate many random trajectories and select those to use as initial guesses for the optimizers through trajectory cost evaluation and collision checks.
3) *Batch Trajectory Optimizer:* Run safety-critical trajectory optimization in batches for each of the selected seed trajectories. Select the best trajectory and pass the corresponding control input to the vehicle.

II. Trajectory Optimization Problem

A. Constrained Optimization Problem

First, let us define the general trajectory optimization problem that we address in this letter. A discrete-time trajectory optimization problem with control input constraints and safety constraints can be defined as follows:

Problem 1 (Constrained Motion Planning):

$$\min_{\boldsymbol{X},\boldsymbol{U}} \quad J(x_0,\boldsymbol{U}) = \sum_{i=0}^{N-1} l^i(x_i,u_i) + l^N(x_N) \tag{1a}$$

$$\text{subject to} \quad x_{i+1} = f(x_i,u_i); \quad i=0,1,\ldots,N-1 \tag{1b}$$

$$u_i^L \le u_i \le u_i^U; \quad i=0,1,\ldots,N-1 \tag{1c}$$

$$d_c(x_i) > r; \quad c=1,\ldots,N_c \tag{1d}$$

where $x_i \in \mathbb{R}^n$ denotes the state and $u_i \in \mathbb{R}^m$ denotes the control input for each time step. We set the control input constraints of (1c) as shown below so that both the magnitude limit u_{lim} and the rate limit $\dot{u}_{lim}$ are accounted for.

$$u_i^U = \min(u_{lim},\ u_{i-1} + \dot{u}_{lim}\Delta t) \tag{2a}$$

$$u_i^L = \max(-u_{lim},\ u_{i-1} - \dot{u}_{lim}\Delta t) \tag{2b}$$

Finally, the inequality (1d) corresponds to the safety constraint. In this letter, we extend the geometry of the vehicle to $N_c = 3$ circles of the same size, as shown in Fig. 2. Thus, constraint (1d) implies that the minimum distance $d_c(x_i)$ from the center of each circle to the unsafe region must be greater than its radius r. The number of circles was determined by balancing the computational cost of checking constraints with a more accurate approximation of the chassis of the ego vehicle.

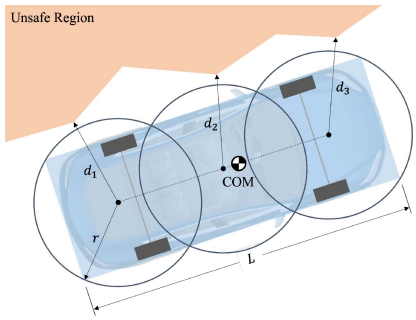


Fig. 2. Notation for the definition of the safety conditions. The vehicle chassis is covered by three circles. The safety condition indicates that the unsafe region must be outside all the circles.

B. Safety-Critical Motion Planning Problem With DBaS

When configuring a trajectory optimization framework, it is most important to strike the right balance between real-time performance and ensuring safety. A common practice here is to replace an unwieldy constraint with a more manageable one, such as changing it to a cost. This not only simplifies the problem but also allows for gradual risk assessment based on the relative position to the obstacle.

In this letter, we redefine the problem by introducing discrete barrier states (DBaSs) [21], which have been frequently used in recent safety-critical motion planning techniques. These states can help synthesize trajectories more efficiently and safely than can recent methods utilizing control barrier functions (CBFs) [22].

We aim to unify the safety constraints (1d) through a single DBaS. Consequently, we define DBaS w_i from the generalized inequality constraint $h_c(x_i)$ as follows.

$$h_c(x_i) = d_c(x_i) - r > 0, \quad c=1,\ldots,N_c \tag{3a}$$

$$w_{i+1} = \sum_{c=1}^{N_c} B(h_c(f(x_i,u_i))) - \beta_c^{des} \tag{3b}$$

Here, the barrier function $\beta_c(x_i) = B(h_c(x_i))$ is chosen to resemble the softplus function [23], [24], with $B(h_c(x_i)) = \frac{\mu}{M}\log(e^{-M\cdot h_c(x_i)}+1)$. With this definition, the desired value of the barrier function is $\beta_c^{des} = 0$, which is the value when

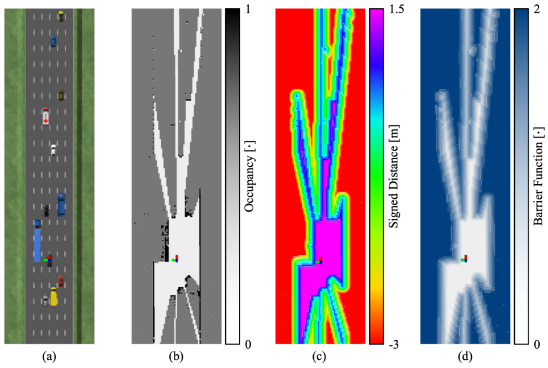


Fig. 4. Example of step-by-step results from the costmap builder: (a) original scene, (b) occupancy grid obtained from the LIDAR point cloud, (c) signed distance field, (d) barrier function map. The values are clipped for visual clarity.

$d_c(x) \to +\infty$. As μ gets larger, the trajectory optimizer pushes the output trajectory more strongly into the safe region, but it is more prone to ill-conditioning. Also, we have $B(h_c(x)) \to \mu \max(-h_c(x), 0)$ as $M \to +\infty$.

Based on the above definition, the safety-critical motion planning problem that we aim to solve with our proposed framework is defined as follows. The augmented state is defined as $\hat{x}_i = [x_i \;\; w_i]^\top$.

Problem 2 (Safety-Critical Motion Planning):

$$\min_{\boldsymbol{X},\boldsymbol{U}} \quad \mathbb{J}(\hat{x}_0, \boldsymbol{U}) = \sum_{i=0}^{N-1} L^i(\hat{x}_i, u_i) + L^N(\hat{x}_N) \tag{4a}$$

$$L^i(\hat{x}_i, u_i) = l^i(x_i, u_i) + \|w_i\|^2_{Q_w} \tag{4b}$$

$$L^N(\hat{x}_N) = l^N(x_N) + \|w_N\|^2_{Q_w} \tag{4c}$$

$$\text{subject to} \quad \hat{x}_{i+1} = F(\hat{x}_i, u_i); \quad i = 0, 1, \ldots, N-1 \tag{4d}$$

$$u_i^L \le u_i \le u_i^U; \quad i = 0, 1, \ldots, N-1 \tag{4e}$$

Based on this problem definition, in the following sections, we will leverage DBaSs across the three phases of the proposed framework. First, the costmap builder precalculates the DBaS in grid notation $\beta[\cdot,\cdot]$. Second, the seed trajectory generator performs a collision check on the candidate trajectories based on the DBaS. Finally, we perform trajectory optimization in batches including the DBaS.

III. Parallel Trajectory Optimization

This section describes the three steps of the proposed parallelized MPC framework via a GPU. Every procedure in the framework was tested on NVIDIA Jetson AGX Orin and simulated in CUDA [25]. Each of these is designed to run within a single time step when all three phases of the process are executed independently with given computing resource. The overall hierearchy of the proposed framework is shown in Fig. 1, and the step-by-step visualizations are shown in Fig. 4.

A. Costmap Builder

Complex environments often limit cognitive capabilities, and it is important to be able to represent the environment in a responsive way. We address this problem using grid notation [20], which can be obtained directly from the raw data of the sensors.

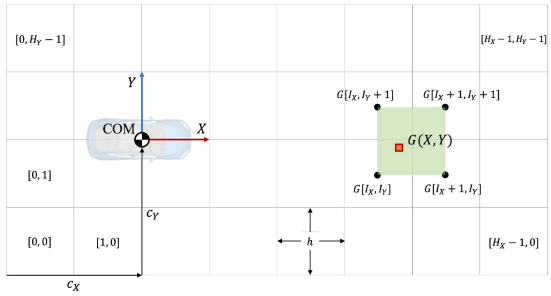


Fig. 3. Notation for the bilinear interpolation of grid cell values. The grid is obtained based on body-fixed coordinates. $(\cdot,\cdot)$ represents the Cartesian coordinate relative to the vehicle, and $[\cdot,\cdot]$ represents the index. Here, $G[\cdot,\cdot]$ is treated as the value at the center position of the corresponding cell.

As an example, we aim to obtain a costmap directly from LIDAR points, assuming no postprocessing capability such as object classification is available. If a higher-level detection capability is available, we can simply "draw" such information onto the grid.

However, the grid notation is spatially discrete, whereas gradient-based MPC requires all variables to be differentiable in the continuous space. Therefore, we utilize a bilinear approximation so that the grid notation can also be utilized in a continuous space. For an arbitrary grid G, the grid cell value at the Cartesian coordinate (X, Y) is obtained from the grid cell values of the four neighboring indices as follows. See Fig. 3 for detailed descriptions.

$$G(X,Y) \approx \begin{bmatrix} 1-R_X & R_X \end{bmatrix} \begin{bmatrix} G[I_X, I_Y] & G[I_X, I_Y+1] \\ G[I_X+1, I_Y] & G[I_X+1, I_Y+1] \end{bmatrix} \begin{bmatrix} 1-R_Y \\ R_Y \end{bmatrix} \tag{5a}$$

$$I_X = \lfloor (X + c_X)/h \rfloor, \; I_Y = \lfloor (Y + c_Y)/h \rfloor, \tag{5b}$$

$$R_X = (X + c_X)/h - I_X, \; R_Y = (Y + c_Y)/h - I_Y. \tag{5c}$$

Here, the $\lfloor \cdot \rfloor$ refers to the floor function. This approximation leads to a smooth continuum of values between grid cells, as opposed to the original discrete definition where all values within same grid cell area have the same value. This allows the entire grid to be approached as a single continuous function in all three of the proposed processes, especially in trajectory optimization.

In this work, we utilize an occupancy grid obtained directly from a LIDAR point cloud as the source of the costmap builder. First, the ground points and the points above the head are removed from the point cloud, and the remaining points are projected onto the grid so that the corresponding grid cells are marked as fully occupied. Then, the occluded cells are considered to have half occupancy, using Bresenham's line algorithm [26] iterating from the center of the body-fixed coordinate system.

The next step is to obtain the signed distance field (SDF) [27] $d[\cdot,\cdot]$ to extract the distance for (1d). Here, we treat a region as unsafe when the occupancy value is above a certain threshold O_{th}. Then, we find the signed distance from each cell to the boundary of the safe/unsafe region, which is positive if it is in the safe region.

The final step is to convert the distance field to a barrier function map $\beta[\cdot,\cdot]$. This can be easily achieved by applying

Algorithm 1: Grid Costmap Generation.

Input: LIDAR points
Output: Occupancy Grid $O[\cdot,\cdot]$, Signed Distance Field $d[\cdot,\cdot]$, Barrier Function Map $\beta[\cdot,\cdot]$
for all *LIDAR points* **do in parallel**
  if *point* **not** *ground* **or** *above head* **then**
    $[I_X, I_Y] \leftarrow$ (5b), $O[I_X, I_Y] \leftarrow 1$
for all *cell indices* $[I_X, I_Y]$ **do in parallel**
  for $[i_X, i_Y]$ **in** `Bresenham` $\{(c_X, c_Y)$ **to** $[i_X, i_Y]\}$ **do**
    $O[I_X, I_Y] \leftarrow \max\{0.5O[i_X, i_Y], O[I_X, I_Y]\}$
  if $O[I_X, I_Y] < O_{th}$ **then**
    Search for $O[i_X, i_Y] \geq O_{th}$
    $d[I_X, I_Y] \leftarrow \min_{[i_X, i_Y]}\{$distance from $[I_X, I_Y]$ to the border of cell $[i_X, i_Y]\}$
  else
    Search for $O[i_X, i_Y] < O_{th}$
    $d[I_X, I_Y] \leftarrow -\min_{[i_X, i_Y]}\{$distance from $[I_X, I_Y]$ to the border of cell $[i_X, i_Y]\}$
  $\beta[I_X, I_Y] \leftarrow a_1 \log(1 + e^{-a_2(d[I_X, I_Y] - r))})$

Algorithm 2: Seed Trajectory Generation.

Input: Initial State x_0, Initial Input Sequence $\bar{\boldsymbol{U}}$, Previous Optimization Result $\hat{\Theta}_{0:S-1}$, Barrier Function Map $\beta(\cdot,\cdot)$
Output: Seed Trajectories $\Theta_{0:S-1}$
for all θ_j **in** $\theta_{0:K-1}$ **do in parallel**
  initial state $x_{0,j} \leftarrow x_0$, $M \gg 1$
  $A \leftarrow f_x(x_0, u_0)$, $B \leftarrow f_u(x_0, u_0)$,
  $\kappa \leftarrow$ DiscreteLQR(A, B, Q, R)
  for $i = 0 : N-1$ **do**
    if $j < S$ **then**
      $\theta_j \leftarrow \hat{\Theta}_j$ with shift as the stamped time difference
    else
      $u^L_{i,j}, u^U_{i,j} \leftarrow$ (2), $u_{i,j} \sim$ (6) or (7)
    $x_{i+1,j} \leftarrow f(x_{i,j}, u_{i,j})$
  Compute trajectory cost $\mathbb{J}_j(\hat{x}_0, \mathbf{U}_j)$
Sort $\theta_{0:K-1}$ with respect to $\mathbb{J}_j(\hat{x}_0, \mathbf{U}_j)$
$\Theta_{0:S-1} \leftarrow \theta_{0:S-1}$

the barrier function defined in the previous section to every grid cell. The resulting barrier function maps can be used directly in the trajectory evaluation and in the trajectory optimizers in later steps.

All of the above processes are parallelized cell-by-cell on the GPU, and the overall process is summarized in Algorithm 1. The size of the grid was determined to allow the designed costmap builder to run at the fastest speed among the three proposed steps. In this letter, we use a grid with 256×128 cells and a cell size of 0.5 meters, and the whole process takes 8.13 ± 1.32 ms with our computing resources. This sequence of steps in the costmap builder significantly reduces the computational burden on the trajectory optimization by precalculating nonlinear barriers that need to be calculated during the iteration. Through the proposed costmap builder, it is possible to visualize the spatial risk assessment from the solver's point of view, making it easy to tune the behavior of the vehicle.

B. Seed Trajectory Generator

Providing a good initial guess is very important for ensuring proper convergence in the trajectory optimization process. The advantage of utilizing GPUs in this regard is the ability to quickly forward-simulate a very large number of trajectories in advance. Therefore, the seed trajectory generator constructed in this study aims to pretest many random trajectories and select a small number of the best trajectories to send to the solver. Since the way we generate the initial guess here can also affect the outcome of the final motion plan, we tested the guesses with two different sampling schemes.

1) Maximum-Range Sampling: In complex driving situations, it is difficult to determine which trajectory will be a good initial guess. Therefore, we set the recursive sampling distribution of the control input sequences to cover the entire range of constraint (2) as the following uniform distribution. This can be seen as part of the process of sampling-based reachability analysis [28]. When using sampling as described below, random set theory suggests that as the number of samples increases, the region covered by the generated trajectory converges to the true reachable set [29]. This means that this form of sampling will theoretically provide the widest and most diverse range of trajectories.

$$u_i \sim \mathcal{U}(u_i^L, u_i^U) \tag{6}$$

2) Gaussian Sampling with Feedback: Although the maximum-range sampling provides diverse trajectories, this inevitably results in unnecessary sampling that does not meet the control objectives. Therefore, we generate a control sequence by sampling the feedforward term of the control with a Gaussian distribution and adding a state feedback term towards the reference trajectory. Given that the initial input trajectory $\bar{\mathbf{U}}$ is obtained from the optimization results of the previous time step, we center $\bar{\mathbf{U}}$ on the sampling distribution of the input v_i as follows.

$$u_i = \langle v_i + \kappa(x_{ref,i} - x_i) \rangle, \quad v_i \sim \mathcal{N}(\bar{u}_i, \Sigma_v) \tag{7}$$

Here, $\langle \cdot \rangle$ represents clamping function by control limit (2). We can simply obtain the state feedback gain κ as a discrete-time LQR solution with approximations $A \approx f_x(x_0, u_0)$, $B \approx f_u(x_0, u_0)$, and weight matrices Q, R same as ones used in the batch trajectory optimizer.

Following the above rules, the seed trajectory generator starts by generating K trajectory samples $\theta_{0:K-1}$ and aims to choose S seed trajectories $\Theta_{0:S-1}$, with $K > S$. We include $\hat{\Theta}_{0:S-1}$, the optimization result from the previous time step, in $\theta_{0:K-1}$ to warm-start some of the batch solvers if possible. Then, we compute the trajectory cost for each sample in parallel to choose the lowest-cost trajectories. Here we temporarily set the barrier function with large M to easily filter out unsafe, infeasible trajectories. This enables $B(0) \to 0+$ when $M \gg 1$, making B resemble softmax function. Finally, seed trajectories $\Theta_{0:S-1}$ are chosen as ones with the least trajectory costs.

The overall procedure is shown in Algorithm 2. K and S were chosen via repeated simulation to allow for sufficient forward simulation while still allowing for fast execution within a single time step of the entire algorithm. The whole process takes 10.97 ± 1.50 ms for $K = 4096$ with our computing resources.

Greater diversity in sampled trajectories is essential to allow for a variety of decisions, but unnecessary sampling can lead to excessive thread usage. As shown in Fig. 5, we are able to control the spatial extent of trajectory sampling, which means that it is possible to select a seed trajectory that deviates from the local minima with fewer random trajectories, as shown in Fig. 6.

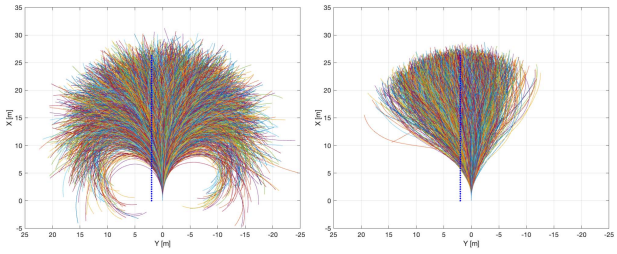


Fig. 5. Comparison between the random trajectories obtained by different sampling strategies: (a) maximum-range sampling and (b) Gaussian sampling with feedback, $\Sigma_v = \text{diag}(7\Delta t[\dot{\delta}_{lim} \;\; \dot{a}_{lim}])^2$. The initial trajectory is given as driving straight at a constant speed of 30km/h. The blue dots represent the reference trajectory.

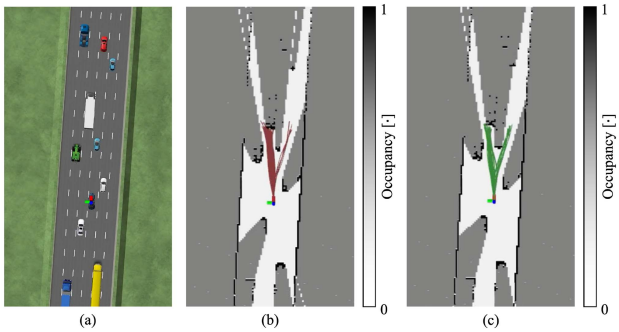


Fig. 6. Comparison between the seed trajectories selected from different types of randomly generated trajectories in the same situation: (a) original scene, (b) maximum-range sampling with $K = 16384$, $S = 128$, (c) Gaussian sampling with feedback with $K = 4096$, $S = 64$, $\Sigma_v = \text{diag}(7\Delta t[\dot{\delta}_{lim} \;\; \dot{a}_{lim}])^2$.

C. Batch Trajectory Optimizer

Finally, the batch trajectory optimizer aims to optimize each of the selected seed trajectories $\Theta_{0:S-1}$ in parallel to obtain refined trajectories $\hat{\Theta}_{0:S-1}$ and select the one with the lowest trajectory cost among them.

For the most computationally demanding trajectory optimization process, we leveraged algorithmic lightweighting and hardware acceleration wherever possible. In this sense, we use a simplified algorithm of differential dynamic programming (DDP) with a DBaS [21] in the form of an iterative linear quadratic regulator (iLQR) to create a real-time optimization process [30]. To this end, we seamlessly combine an algorithmic methodology that allows parallelization within the optimization process while considering input constraints to achieve the highest possible real-time performance [31]. The entire process is summarized in Algorithm 3.

Following the problem definition in Section II, the dynamics of the vehicle are represented by a bicycle model [32]. To satisfy both the accuracy of the model and the real-time usability of MPC, the simplest form of lateral dynamics was used, with longitudinal control being achieved through sub-control of acceleration. The state of the vehicle is defined as $x = [X \;\; Y \;\; \psi \;\; v_x \;\; v_y \;\; r]^\top$, which represents the global position and yaw (X, Y, ψ) of the center of mass and the longitudinal, lateral, and rotational velocities (v_x, v_y, r), respectively. The control input is defined as $u = [\delta \;\; a_x]^\top$, corresponding to the steering angle and longitudinal acceleration, respectively. The detailed description on the notations are shown in Fig. 7.

Algorithm 3: Batch Trajectory Optimization.

Input: Seed Trajectories $\Theta_{0:S-1}$, Barrier Function Map $\beta(\cdot,\cdot)$
Output: Optimization Results $\hat{\Theta}_{0:S-1}$, Best Trajectory Θ^*
for all *seed trajectories* $\Theta_{0:S-1}$ **do in parallel**
  repeat
    for all *derivative types and* $i = 0 : N$ **do in parallel**
      Compute $F^i_{\hat{x}}$, F^i_u, $L^i_{\hat{x}}$, L^i_u, $L^i_{\hat{x}\hat{x}}$, $L^i_{u\hat{x}}$, L^i_{uu}
    $V^N_{\hat{x}} \leftarrow L^N_{\hat{x}}$, $V^N_{\hat{x}\hat{x}} \leftarrow L^N_{\hat{x}\hat{x}}$
    for $i = N-1 : 0$ **do**
      $Q^i_{\hat{x}} \leftarrow L^i_{\hat{x}} + F^{i\top}_{\hat{x}} V^{i+1}_{\hat{x}}$
      $Q^i_u \leftarrow L^i_u + F^{i\top}_u V^{i+1}_{\hat{x}}$
      $Q^i_{\hat{x}\hat{x}} \leftarrow L^i_{\hat{x}\hat{x}} + F^{i\top}_{\hat{x}} V^{i+1}_{\hat{x}\hat{x}} F^i_{\hat{x}}$
      $Q^i_{uu} \leftarrow L^i_{uu} + F^{i\top}_u V^{i+1}_{\hat{x}\hat{x}} F^i_u$
      $Q^i_{u\hat{x}} \leftarrow L^i_{u\hat{x}} + F^{i\top}_u V^{i+1}_{\hat{x}\hat{x}} F^i_{\hat{x}}$
      $k_i \leftarrow \text{argmin}_{\delta u_i} \frac{1}{2}\delta u_i^\top Q^i_{uu}\delta u_i + Q^{i\top}_u \delta u_i$ s.t. (4e)
      $K_i \leftarrow -\tilde{Q}^{-1}_{uu}\tilde{Q}_{u\hat{x}}$ for indices within (4e)
      $V^i_{\hat{x}} \leftarrow Q^i_{\hat{x}} - K_i^\top Q^i_{uu} k_i$, $V^i_{\hat{x}\hat{x}} \leftarrow Q^i_{\hat{x}\hat{x}} - K_i^\top Q^i_{uu} K_i$
    for all *line search indices* ν **do in parallel**
      for $i = 0 : N-1$ **do**
        $u_{i,\nu} \leftarrow u_i + \alpha_\nu k_i + K_i(\hat{x}_{i,\nu} - \hat{x}_i)$
        $\hat{x}_{i+1,\nu} \leftarrow F(\hat{x}_{i,\nu}, u_{i,\nu})$
    $\nu^* \leftarrow \arg\min_\nu \mathbb{J}(\hat{x}_0, \boldsymbol{U}_\nu)$
    $\Theta_s = \{\hat{\boldsymbol{X}}_s, \boldsymbol{U}_s\} \leftarrow \{\hat{\boldsymbol{X}}_\nu, \boldsymbol{U}_\nu\}$, $\hat{\Theta}_s \leftarrow \Theta_s$
  until *convergence or maximum number of iterations*
$s^* \leftarrow \arg\min_s \mathbb{J}(\hat{x}_0, \boldsymbol{U}_s)$, $\Theta^* \leftarrow \Theta_{s^*}$

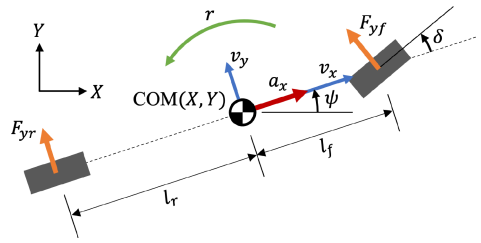


Fig. 7. Notation for the dynamic bicycle model of the vehicle.

$$\dot{x} = f_c(x,u) = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\psi} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{r} \end{bmatrix} = \begin{bmatrix} v_x\cos\psi - v_y\sin\psi \\ v_x\sin\psi + v_y\cos\psi \\ r \\ a_x \\ \frac{1}{m}(F_{yr} + F_{yf}\cos\delta - m v_x r) \\ \frac{1}{I_z}(F_{yf} l_f \cos\delta - F_{yr} l_r) \end{bmatrix} \tag{8a}$$

$$x_{i+1} = f(x_i, u_i) = x_i + f_c(x_i, u_i)\Delta t \tag{8b}$$

The lateral forces F_{yf} and F_{yr} from the tire are simply modeled as follows using the cornering stiffness C_f, C_r [33].

$$\alpha_f = \delta - \tan^{-1}\left(\frac{l_f r + v_y}{v_x}\right), \qquad F_{yf} = C_f \alpha_f \tag{9a}$$

$$\alpha_r = \tan^{-1}\left(\frac{l_r r - v_y}{v_x}\right), \qquad F_{yr} = C_r \alpha_r \tag{9b}$$

The cost function is designed to follow the reference trajectory $\{\mathbf{X}_{ref}, \mathbf{Y}_{ref}\}$.

$$l^i(x_i, u_i) = Q_d\left[(X_i - X_{i,ref})^2 + (Y_i - Y_{i,ref})^2\right] + \|u_i\|^2_R \tag{10}$$

Each of the batch solver aims to solve Problem 2. By definition (10), the ego vehicle follows the reference trajectory, and the safety conditions are reflected in the DBaS cost in definition (4c) to avoid obstacles. Here, DBaS is obtained directly from the

TABLE I
PARAMETER SPECIFICATIONS

Grid Costmap Generation			
μ, M	0.5, 5	O_{th}	0.1
H_X, H_Y	256, 128	h	0.5m
c_X, c_Y	32m, 32m		
Model Parameters			
l_f, l_r	1.45m, 1.52 m	m, I_z	920.93kg, 1585.3kg·m^2
C_f, C_r	97.3kN/rad, 97.3kN/rad		
Problem Settings			
N, Δt	63, 0.05s	Q_d, R	0.01, diag([10 0.1])
δ_{lim}	0.6467rad	$a_{x,lim}$	2m/s^2
$\dot{\delta}_{lim}$	1.2934rad/s	$\dot{a}_{x,lim}$	4m/s^3

TABLE II
SCENARIO DESCRIPTIONS

	Ego Target Velocity	Objects Case 1	Objects Case 2
Scenario 1	30km/h	10km/h, 15%	0km/h, 20%
Scenario 2	50km/h	25km/h, 10%	15km/h, 15%
Scenario 3	70km/h	40km/h, 5%	30km/h, 10%

Each object description represents: object velocity [km/h], traffic density (total length occupied / road length) [%].

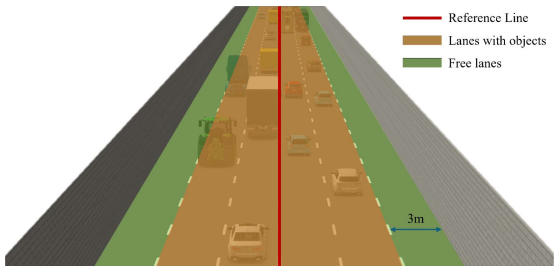


Fig. 8. Simulation environment descriptions for random object avoidance scenarios.

barrier function map $\beta[\cdot,\cdot]$, which is the output of the costmap builder.

Technically, each solver is implemented in a single CUDA thread block, so all solvers run in parallel on a block-by-block basis. Parallelization in the solver is performed within thread blocks, and variables within the solver are stored in CUDA's shared memory to achieve maximum read/write speeds. After applying all of the real-time performance considerations listed above, on our test hardware, the optimization of all seed trajectories takes 26.72± 2.94ms, so it can be run stably within the control period of 50ms.

IV. EXPERIMENTS AND RESULTS

To evaluate the practical usability of the proposed algorithm, a real-time simulation system was constructed using CarMaker [34], a widely used realistic vehicle simulator. The system consisted of a PC running the control algorithm and a PC running CarMaker, which communicated through the Robotic Operating System (ROS). The control PC was NVDIA Jetson AGX Orin, which ran the costmap builder and seed trajectory generator at 100 Hz maximum and the batch trajectory optimizer at 20 Hz. The simulation PC reported the vehicle status at a rate of 100 Hz and simulated the Ouster OS1-64 LIDAR sensor mounted on top of the vehicle.

The ego vehicle drove along the center of the road and avoided various types of random objects that appeared in the middle four lanes out of six lanes, as shown in Fig. 8.The reference trajectory was given as $\{X_{i,ref}, Y_{i,ref}\} = \{X + v_t i\Delta t, 0\}$ to follow the target velocity v_t and the center of the road. Each lane was 3 meters wide, and the test road was a straight line with a total length of 1 km. Fifty percent of the objects were passenger vehicles, and the other vehicles were trucks, motorcycles, etc. Three main scenarios based on the speed of the ego vehicle were prepared, and each scenario was prepared for two cases: one with high relative speed and traffic density, and the other with low relative speed and traffic density. Each case was prepared with 20 runs with different configurations of randomly generated objects. The details on the physical parameters are shown in Table I, and the detailed scenario settings are shown in Table II.

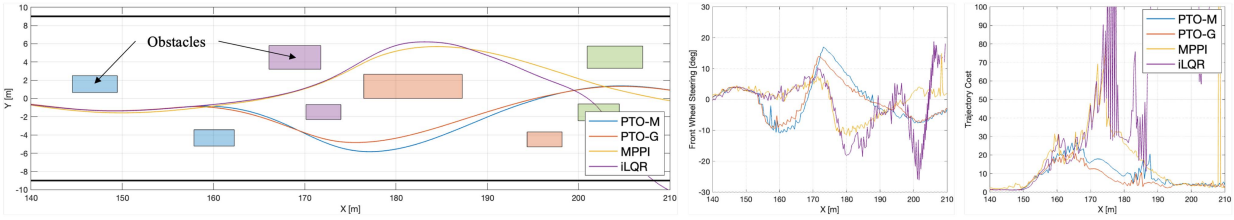
TABLE III
SUMMARY OF RANDOM OBJECT AVOIDANCE SCENARIOS

	Scenario 1							
	Objects Case 1				Objects Case 2			
Items	PTO-M	PTO-G	MPPI	iLQR	PTO-M	PTO-G	MPPI	iLQR
e_{lat}^{rms} [m]	1.951	**1.917**	2.199	2.232	**1.871**	1.926	2.332	2.606
e_{v}^{rms} [m/s]	**0.374**	0.377	0.671	1.003	0.429	**0.393**	0.708	0.898
$\dot{\delta}^{rms}$ [rad/s]	0.380	**0.353**	0.747	0.863	0.232	**0.224**	0.561	1.091
$\dot{a}^{rms}$ [m/s^3]	**0.974**	1.080	4.765	1.911	1.091	**1.035**	4.278	1.811
N_{com}	**20/20**	**20/20**	18/20	11/20	**20/20**	**20/20**	18/20	4/20
N_{col}	**0**	**0**	2	21	**0**	**0**	10	23
	Scenario 2							
	Objects Case 1				Objects Case 2			
Items	PTO-M	PTO-G	MPPI	iLQR	PTO-M	PTO-G	MPPI	iLQR
e_{lat}^{rms} [m]	1.730	**1.705**	2.145	2.221	1.998	**1.956**	2.231	2.541
e_{v}^{rms} [m/s]	0.330	**0.318**	0.694	1.717	**0.310**	0.369	0.779	1.851
$\dot{\delta}^{rms}$ [rad/s]	**0.368**	0.387	0.741	0.647	**0.311**	0.388	0.743	0.652
$\dot{a}^{rms}$ [m/s^3]	**0.735**	0.788	6.111	2.012	**1.259**	1.277	5.810	1.983
N_{com}	**20/20**	**20/20**	20/20	15/20	**20/20**	**20/20**	18/20	5/20
N_{col}	**0**	**0**	0	10	**0**	**0**	7	15
	Scenario 3							
	Objects Case 1				Objects Case 2			
Items	PTO-M	PTO-G	MPPI	iLQR	PTO-M	PTO-G	MPPI	iLQR
e_{lat}^{rms} [m]	**1.043**	1.081	1.581	1.515	1.122	**1.090**	2.290	2.350
e_{v}^{rms} [m/s]	**0.181**	0.199	1.220	1.190	0.259	**0.231**	1.131	1.101
$\dot{\delta}^{rms}$ [rad/s]	**0.214**	0.216	1.213	0.640	0.233	**0.209**	1.099	0.531
$\dot{a}^{rms}$ [m/s^3]	**0.540**	0.588	9.111	2.103	**0.571**	0.605	9.909	2.115
N_{com}	**20/20**	**20/20**	19/20	18/20	**20/20**	**20/20**	19/20	14/20
N_{col}	**0**	**0**	1	3	**0**	**0**	3	9

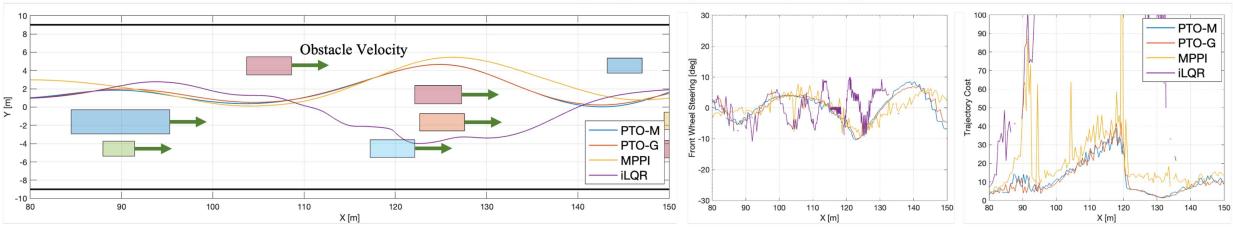
In this section, we compare four control methods: the proposed parallel trajectory optimization framework with max-range sampling (PTO-M) and Gaussian sampling with feedback (PTO-G) as random input sampling methods, and MPPI and iLQR, which optimize only one trajectory. All the kinds of solvers share the same problem definition, which is defined in Problem 2. We use $K = 16384$ and $S = 128$ for PTO-M, and $K = 4096$, $S = 64$ and $\Sigma_v = \text{diag}(7\Delta t[\dot{\delta}_{lim} \;\; \dot{a}_{lim}])^2$ for PTO-G. For MPPI, for practical application, we apply the Savitzky–Golay filter [11] and the constraint (2) on the input sequence at the end of every iteration. The covariance for control input sampling for MPPI is set to $\Sigma_u = \text{diag}(3\Delta t[\dot{\delta}_{lim} \;\; \dot{a}_{lim}])^2$. For normal iLQR, we implement the proposed framework by limiting the number of seed trajectories to one.

The assessment metrics are the RMS values of the lateral error e_{lat} from the center of the road, the error e_v from the target speed, the rate of change in the steering input $\dot{\delta} \approx \Delta\delta/\Delta t$, and the rate of change in the acceleration input $\dot{a} \approx \Delta a/\Delta t$. For each scenario, we also observed the number of cases N_{com} that were fully run to the end of the scenario out of the 20 runs, as well as the total number of collisions N_{col} during the scenario.

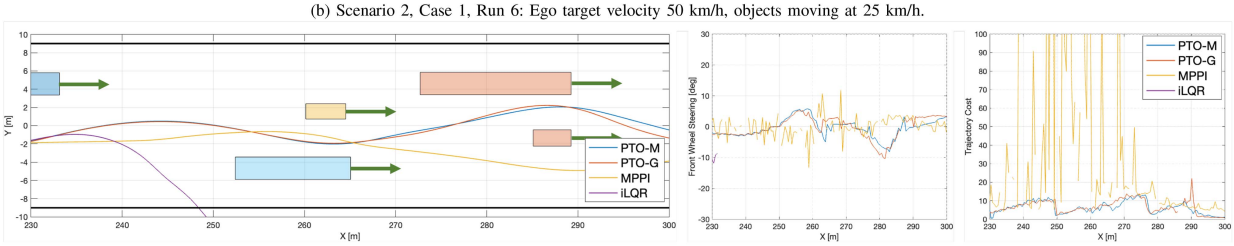
The comprehensive quantified outcomes are summarized in Table III. Overall, we can see that the proposed framework is able to safely perform all the given scenarios. In contrast, MPPI caused some crashes or failed to complete the scenarios, and iLQR could not handle most complex environments. In addition, the proposed PTO-M and PTO-G outperformed MPPI in terms of all the evaluation metrics. In particular, they show a larger gap in terms of $\dot{\delta}^{rms}$ and $\dot{a}^{rms}$, indicating an improvement in the accuracy of control. This can be explained by the successful



(a) Scenario 1, Case 2, Run 3: Ego target velocity 30 km/h, objects stopped.



(b) Scenario 2, Case 1, Run 6: Ego target velocity 50 km/h, objects moving at 25 km/h.



(c) Scenario 3, Case 2, Run 7: Ego target velocity 70 km/h, Objects moving at 30 km/h.

Fig. 9. Motion results in several randomly generated cases. All motion history is shown relative to the objects for ease of viewing. Each case represents the ego motion using different control methods, the generated steering input, and the cost of the planned trajectory. In (c), iLQR failed before reaching the scene shown in the figure. For detailed motion in real time, see Video 1.

combination of responsiveness to various situations due to random trajectory sampling and the advantages of gradient-based MPC in providing high-quality optimization results, which was the goal of this study.

Since both PTO-M and PTO-G ultimately use the same form of gradient-based optimization, we can see that they result in similar motion, ensuring safety. Here, it is worth noting that PTO-G produces results similar to those of PTO-M while using fewer trajectories. These results suggest that the number of threads required can be reduced by choosing the appropriate random trajectory generation rules.

Notable motion results for different situations under each control method are shown in Fig. 9. In Fig. 9(a), avoiding obstacles placed in a small pocket-shaped group near $X = 170$ m to $X = 180$ m requires a decision that completely exceeds the local minima of the costmap. We can observe that this behavior is performed safely when utilizing the proposed control framework but fails in the case of MPPI and regular iLQR. It can also be seen that the costs of the trajectory planned by PTO-G and PTO-M are lowest among all control methods, successfully choosing the best trajectory among multiple local minima. These results confirms that in practice, proper selection of trajectory synthesis method such as from PTO-G could be effective in saving the number of threads where the available number of threads is limited.

It can also be seen that the proposed control framework finds the safe trajectory with the least cost compared to that of MPPI and results in safe driving, not only in the low-speed environment of scenario 1 (Fig. 9(a)) but also in an environment with higher ego target velocity and object velocity, as in scenario 2 (Fig. 9(b)) and scenario 3 (Fig. 9(c)).

V. Conclusion and Discussion

In this letter, we introduce a control framework that combines the advantages of random trajectory sampling with the advantages of gradient-based trajectory optimization through parallel execution of MPC solvers. First, we formulate a generalized safety-embedded MPC problem using DBaSs. The proposed control framework has three phases. In the first step, the costmap builder, a barrier function map, is derived from the raw occupancy grid map. In the second step, the seed trajectory generator randomly samples many trajectories and selects the seed trajectories to send to the batch optimizer. The final step, the batch trajectory optimizer, runs iLQR with a DBaS in parallel and sends the best trajectory and the corresponding control sequence to the controller.

In our real-time simulation experiments, we identified several advantages of the proposed control framework. We found that the proposed framework is suitable for generating safe collision-free trajectories in complex urban environments since it considers various possibilities rather than single trajectory optimization. In addition, unlike MPPI, it uses gradient-based optimization, which enables more accurate control actions.

Meanwhile, the grid size, number of initial trajectory samples, and number of seed trajectories should be carefully set considering available threads and computing capability. For use with other algorithms in autonomous driving, we may need to have dedicated threads or a separate parallel processing unit for this to ensure a sufficient number of trajectory simulations.

In this study, we used only the raw occupancy grid as the source of the costmap, assuming that the detection capability is weak. However, in the future, when detection can be performed,

we plan to incorporate this information as a source of the costmap. Specifically, we aim to directly evaluate the risk level of each grid cell through reachability analysis for the objects and utilize it as a safety condition in addition to raw occupancy.

REFERENCES

[1] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser, "A review of motion planning for highway autonomous driving," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 5, pp. 1826–1848, May 2020.

[2] S. Teng et al., "Motion planning for autonomous driving: The state of the art and future perspectives," *IEEE Trans. Intell. Veh.*, vol. 8, no. 6, pp. 3692–3711, Jun. 2023.

[3] S. Han, J. Kwon, and K.-S. Kim, "Position uncertainty-integrated potential function for collision avoidance systems based on model predictive control," *IEEE Trans. Intell. Veh.*, early access, Apr. 24, 2024, doi: 10.1109/TIV.2024.3392895.

[4] Y. Rasekhipour, A. Khajepour, S.-K. Chen, and B. Litkouhi, "A potential field-based model predictive path-planning controller for autonomous road vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 5, pp. 1255–1267, May 2017.

[5] M. Brown and J. C. Gerdes, "Coordinating tire forces to avoid obstacles using nonlinear model predictive control," *IEEE Trans. Intell. Veh.*, vol. 5, no. 1, pp. 21–31, Mar. 2020.

[6] P. E. Gill and E. Wong, "Sequential quadratic programming methods," in *Proc. Mixed Integer Nonlinear Program.*, 2011, pp. 147–224.

[7] D. Murray and S. Yakowitz, "Differential dynamic programming and Newton's method for discrete optimal control problems," *J. Optim. Theory Appl.*, vol. 43, no. 3, pp. 395–414, 1984.

[8] T. Brüdigam, F. Di Luzio, L. Pallottino, D. Wollherr, and M. Leibold, "Grid-based stochastic model predictive control for trajectory planning in uncertain environments," in *Proc. IEEE 23 rd Int. Conf. Intell. Transp. Syst.*, 2020, pp. 1–8.

[9] J. Liu, P. Jayakumar, J. L. Stein, and T. Ersal, "A nonlinear model predictive control formulation for obstacle avoidance in high-speed autonomous ground vehicles in unstructured environments," *Veh. Syst. Dyn.*, vol. 56, no. 6, pp. 853–882, 2018.

[10] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2016, pp. 1433–1440.

[11] 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 Trans. Robot.*, vol. 34, no. 6, pp. 1603–1622, Dec. 2018.

[12] I. S. Mohamed, K. Yin, and L. Liu, "Autonomous navigation of AGVs in unknown cluttered environments: Log-MPPI control strategy," *IEEE Robot. Automat. Lett.*, vol. 7, no. 4, pp. 10240–10247, Oct. 2022.

[13] G. Williams, B. Goldfain, P. Drews, K. Saigol, J. M. Rehg, and E. A. Theodorou, "Robust sampling based model predictive control with sparse objective information," in *Proc. Robot.: Sci. Syst.*, 2018, vol. 14, p. 2018.

[14] J. Yin, Z. Zhang, E. Theodorou, and P. Tsiotras, "Trajectory distribution control for model predictive path integral control using covariance steering," in *Proc. Int. Conf. Robot. Automat.*, 2022, pp. 1478–1484.

[15] Z. Ugray, L. Lasdon, J. Plummer, F. Glover, J. Kelly, and R. Martí, "Scatter search and local NLP solvers: A multistart framework for global optimization," *Informs J. Comput.*, vol. 19, no. 3, pp. 328–340, 2007.

[16] R. Martí, J. A. Lozano, A. Mendiburu, and L. Hernando, "Multi-start methods," in *Handbook Heuristics*. Berlin, Germany: Springer, 2018, pp. 155–175.

[17] A. Rinnooy Kan and G. Timmer, "Stochastic global optimization methods part ii: Multi level methods," *Math. Program.*, vol. 39, pp. 57–78, 1987.

[18] G. L. Nemhauser, A. R. Kan, and M. Todd, *Handbooks in Operations Research and Management Science*. Amsterdam, the Netherlands: North-Holland, 1989.

[19] T. Van Luong, N. Melab, and E.-G. Talbi, "Gpu-based multi-start local search algorithms," in *Proc. Learn. Intell. Optimization: 5th Int. Conf.*, 2011, pp. 321–335.

[20] M. Saval-Calvo, L. Medina-Valdés, J. M. Castillo-Secilla, S. Cuenca-Asensi, A. Martínez-Álvarez, and J. Villagrá, "A review of the Bayesian occupancy filter," *Sensors*, vol. 17, no. 2, 2017, Art. no. 344.

[21] H. Almubarak, K. Stachowicz, N. Sadegh, and E. A. Theodorou, "Safety embedded differential dynamic programming using discrete barrier states," *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 2755–2762, Apr. 2022.

[22] A. J. Taylor and A. D. Ames, "Adaptive safety with control barrier functions," in *Proc. Amer. Control Conf.*, 2020, pp. 1399–1405.

[23] T. Szandała, "Review and comparison of commonly used activation functions for deep neural networks," in *Bio-Inspired Neurocomputing*. Berlin, Germany: Springer, 2021, pp. 203–224.

[24] Y. Guo, D. Yao, B. Li, Z. He, H. Gao, and L. Li, "Trajectory planning for an autonomous vehicle in spatially constrained environments," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 10, pp. 18326–18336, Oct. 2022.

[25] P. N. Vingelmann and F. H. Fitzek, "Cuda, release: 10.2.89," 2020. [Online]. Available: https://developer.nvidia.com/cuda-toolkit

[26] P. Koopman, "Bresenham line-drawing algorithm," *Forth Dimens.*, vol. 8, no. 6, pp. 12–16, 1987.

[27] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart, "Signed distance fields: A natural representation for both mapping and planning," in *Proc. RSS 2016 Workshop: Geometry Beyond-representations, Phys., Scene Understanding Robot.*, 2016, doi: 10.3929/ethz-a-010820134.

[28] T. Lew and M. Pavone, "Sampling-based reachability analysis: A random set theory approach with adversarial sampling," in *Proc. Conf. Robot Learn.*, 2021, pp. 2055–2070.

[29] T. Lew, L. Janson, R. Bonalli, and M. Pavone, "A simple and efficient sampling-based algorithm for general reachability analysis," in *Proc. Learn. Dyn. Control Conf.*, 2022, pp. 1086–1099.

[30] M. Cho, Y. Lee, and K.-S. Kim, "Model predictive control of autonomous vehicles with integrated barriers using occupancy grid maps," *IEEE Robot. Automat. Lett.*, vol. 8, no. 4, pp. 2006–2013, Apr. 2023.

[31] Y. Lee, M. Cho, and K.-S. Kim, "GPU-parallelized iterative LQR with input constraints for fast collision avoidance of autonomous vehicles," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 4797–4804.

[32] G. Bellegarda and Q. Nguyen, "Dynamic vehicle drifting with nonlinear MPC and a fused kinematic-dynamic bicycle model," *IEEE Control Syst. Lett.*, vol. 6, pp. 1958–1963, 2021.

[33] G. Baffet, A. Charara, and D. Lechner, "Estimation of vehicle sideslip, tire force and wheel cornering stiffness," *Control Eng. Pract.*, vol. 17, no. 11, pp. 1255–1264, 2009.

[34] "Carmaker," IPG Automotive. (n.d.). Oct. 7, 2024. [Online]. Available: https://ipg-automotive.com/