CURVATURE AND ENERGY-BASED TRAJECTORY OPTIMIZATION IN UNSTRUCTURED ENVIRONMENTS

JEONGYONG YANG*, HOJIN JU*, AND SOOJEAN HAN*

Abstract. Safe and efficient path planning is crucial for autonomous driving systems. In this paper, we propose a trajectory optimization algorithm for unstructured environments that minimizes both curvature and energy while achieving a near-time-optimal path. To improve efficiency of the optimization process, we employ path velocity decomposition and reformulate the complex optimization problem with the agent's dynamic constraints into a quadratic programming (QP) problem, considering the agent as a point of mass. Previous curvature-based algorithms were primarily designed for racing scenarios, limited to static bounded tracks. In contrast, our approach extends their applicability to complex open environments with obstacles, enabling safe and efficient navigation through the use of a boundary tube constraint. Furthermore, we introduce several techniques to adapt the algorithm to unstructured environments, such as adaptive path spacing to enhance computational efficiency by reducing the dimension of the QP problem. Additional methods, including boundary tube correction and decaying curvature bound, ensure more feasible and stable solutions for the algorithm. The proposed algorithm was implemented in ROS Gazebo and successfully tested on a four-wheel drive robot. Results demonstrate that the proposed algorithm generates safe, smooth and time efficient paths, showing improved performance compared to B-spline and the original curvature-based optimization method.

Key words. path planning, trajectory planning, path optimization, trajectory optimization, autonomous driving, mobile robot

1. Introduction.

In the modern era of autonomous transportation, path planning is essential for agents to navigate around complex and unstructured environments, which are filled with many obstacles. Traditional path planning methods prioritize metrics such as shortest distance and minimum travel time. While these objectives are critical, it is equally important to achieve smooth trajectories with minimum curvature in order to control an agent stably. Minimizing curvature reduces wear and strain on the mechanical components (e.g., wheels, motors, joints) and extends their lifespan. Moreover, smooth paths enhance energy efficiency by reducing unnecessary acceleration and deceleration, improving passenger comfort and safety.

To address these challenges, this paper introduces a novel curvature and energy-based trajectory optimization framework designed specifically for unstructured environments. By modeling the agent as a point of mass, the complex nonlinear dynamics are simplified into a tractable quadratic programming (QP) formulation. Unlike traditional minimum distance methods such as the RRT family [9, 10, 14, 37] or the A* family [6, 11, 20, 38], our approach focuses on minimizing the path's curvature and energy, achieving a near-time optimal path with low computational cost. Since the optimization can fail in complex environments, we also introduce several techniques to improve stability and efficiency of the optimization.

The remainder of this paper is organized as follows: Section 2 reviews related work, providing an overview of existing trajectory optimization techniques, and Section 3 describes the system model and environment. We present our proposed trajectory optimization framework in Section 4, including several methods for improving safety and efficiency. We demonstrate the performance of our proposed method using a numerical experiment in Section 5, including the specific tracking controller used and a comparison of several performance metrics, including computation time and

*School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, Republic of Korea (seiryu2238@kaist.ac.kr, juhojin@kaist.ac.kr, soojean@kaist.ac.kr).

travel time. Section 6 concludes the paper.

2. Related Work. Path smoothing methods have evolved with classic techniques such as cubic spline, Bézier and B-spline curves, and more advanced constructs like Non-Uniform Rational B-splines (NURBS) [30, 36]. These methods provide various advantages in terms of computational efficiency, trajectory continuity (C^1- or C^2-continuity), and suitability for complex environments. Cubic splines [21] are constructed with piecewise cubic polynomial curves that ensure C^2-continuity with low computational cost. Bézier [5, 34, 39] curves generate smooth paths using control points, offering low computational cost. However, since the curve is affected by all control points, a change in one control point affects the entire curve, and determining optimal control point placement can be challenging. Like Bézier curves, B-splines [2, 16, 26] generate smooth paths using control points, but allow for local control, meaning that changes to a control point only affect a portion of the path. This local control and low computational cost make B-splines advantageous for fast optimization. However, B-splines do not necessarily pass through the control points, which can make it difficult to shape the path as desired, and as the number of control points and the degree of the curve increase, so does the computational cost.

Special curves like Dubins [7, 25] and clothoids [22, 33] are employed for their unique ability to incorporate curvature and smoothness. On one hand, Dubin's curve has low computational load, but it has discontinuous curvature profiles. This discontinuity results in a jerky transition when switching between straight segments and circular arcs, which is not physically achievable by most vehicles. Parlangeli et al. [29] address this limitation by approximating Dubins paths with C^∞ continuous paths, providing a smooth transition for wheeled robots. On the other hand, since the curvature of clothoids changes linearly, it is particularly suitable for high-speed navigation. Thus, it is commonly used in highway road designs. Kim et al. [19] proposed a modified turning algorithm reducing both path's length and curvature, achieving low centrifugal force.

Optimization-based methods mathematically define constraints such as path curvature and length to generate objective-driven trajectories. Techniques like MILP [24], MPC [8, 23, 27, 41] enable the direct creation of smooth paths, considering the agent's dynamics. Since the directly generated trajectories incorporate dynamic constraints, these methods provide realistic trajectory planning. However, they still require significant computational resources despite efforts to reduce the burden. Alternatively, there are methods that optimize pre-existing paths using optimization-based smoothing (e.g., [31]). These smoothing approaches have the advantage of satisfying path constraints while requiring less computational resources.

Optimization-based methods have recently advanced, especially in the field of time-critical racing [32, 40], as smooth trajectories are critical for achieving faster cornering and maintaining consistently high speeds. Kapania et al. [18] introduced a two-step iterative algorithm for generating minimum time trajectories. They decoupled the process into two sequential subproblems: a forward-backward integration scheme to compute a minimum-time velocity profile for a fixed path, and a convex optimization problem to update the path by minimizing its curvature while respecting vehicle dynamics and track constraints. This method can obtain the optimal solution rapidly due to its fast convergence. Braghin et al. [4] used a combined objective function that blends the shortest path and least curvature trajectories, and formulated a QP problem. Then, they obtain the optimized trajectory in an iterative way. They also emphasized the balance between minimizing curvature and distance reduction to achieve the shortest lap times. Heilmeier et al. [12] proposed another QP-based trajectory smoothing method that generates minimum curvature paths with low computational overhead. They formulated iterative QP problem to obtain minimum curvature trajectory, and demonstrated that this approach achieves lap times close to those of a time-optimal method. The two methods differ in their assumptions: the former removes the cross term in the square of curvature, while the latter approximates the first derivative of the path as constant. Both methods use iteration to reduce errors arising from these assumptions. Additionally, Chung et al. [15] employed a geometric weighted sum, combining a minimum curvature trajectory and the shortest path. By adjusting the geometric weight parameter, they demonstrated that this approach effectively reduces lap times under different maximum velocity.

3. System Model and Environment.

3.1. System Model Description. The kinematics or dynamics of the agent are generally described by with system dynamics $\dot{\mathbf{x}} = f(t, \mathbf{x}, \mathbf{u})$, and measurement equation $\mathbf{y} = g(t, \mathbf{x}, \mathbf{u})$, where $\mathbf{x}$ represents the state of the agent and $\mathbf{u}$ represents the control input. The specific f and g functions vary according to the agent type and configuration (see Subsection 5.1 for details).

Remark 3.1 (Agent Type). Although our formulation is exclusively written for wheeled, driving robots (e.g., autonomous vehicles, RC cars) throughout this paper, we emphasize that our algorithm can be extended easily to other types of robots. We will henceforth refer to the "robot" or "vehicle" as "agent".

To separate the agent's dynamics from the trajectory optimization, the agent is treated as a point mass. It eliminates the need for incorporating dynamic constraints into the optimization. Instead, the velocity profile is calculated after the optimization. This path velocity decomposition (PVD) [17] approach simplifies the optimization problem and reduces computational time, as complex dynamic constraints are handled separately.

3.2. Environment Description. Unstructured environments lack predefined or structured features such as roads, lanes, or navigational aids, for examples, off-road terrains, forested areas, or indoor spaces with irregular layouts.

At the beginning of the planning, we assume that the agent has access to a global map of the environment. This map can be derived from prior knowledge or obtained through pre-mapping processes, enabling the agent to plan its trajectory effectively within the unstructured space.

4. Proposed Trajectory Optimization Method.

4.1. Overall Structure. Figure 1 illustrates the structure of the proposed trajectory optimization method. The initial path from $\mathbf{x}_{\text{start}}$ to $\mathbf{x}_{\text{goal}}$ is determined using sampling-based methods such as the RRT* family, or graph-based methods such as the A* family. These methods not only compute a feasible route but also define the homotopy class of the optimized path. We remark that this paper does not focus on optimization methods for static path-planning; instead, it aims to find an optimal path within a given homotopy class. Thus, any method can be used to generate the initial path and our methods will still work.

The iterative QP process begins to optimize the initial path. To improve computational efficiency, adaptive spacing is applied to the path to reduce the number of path points while maintaining sufficient resolution for optimization. This is followed by a boundary tube generation that serves as constraints for the optimization. The

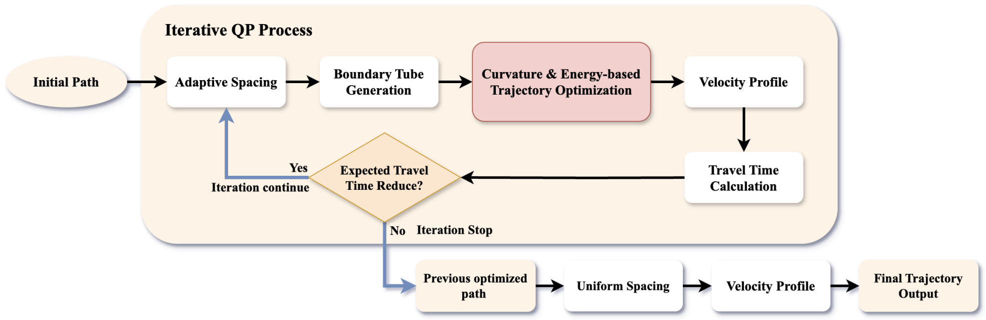


Fig. 1: Total trajectory optimization process

objective function in the QP includes the curvature and the energy cost of the path. At each iteration, the velocity profile is calculated to estimate the travel time based on the optimized path. The process continues iteratively until the expected travel time no longer decreases. The method finally selects the path with the minimum expected travel time, uniformly spaces it for lateral control, and generates a velocity profile for longitudinal control.

One advantage of using the iterative method is that even if the process is interrupted, the agent can still make use of the partially optimized path. This flexibility is especially useful for real-time applications where a fully converged solution may not always be available. Also, when used in an online setting, the agent might initially follow a non-smooth path, but as the iterations progress, the path will gradually become smoother and closer to optimal path. Eventually, the agent will follow the fully optimized path.

4.2. Path Representation. As shown in Figure 2, the path points are adjusted along the normal direction of each path point, as defined below:

$$\mathbf{s}_i^{\text{new}} = \mathbf{s}_i + \alpha_i \mathbf{n}_i \tag{4.1}$$

where $\mathbf{s}_i^{\text{new}}$ denotes the new path point, $\mathbf{s}_i$ is the previous path point, α_i is the deviation along the normal direction, and $\mathbf{n}_i$ represents the normalized normal vector of the i-th path point. The deviation α_i is constrained by the left and right boundaries to prevent obstacle-collisions as follows:

$$-d_{i,L} \leq \alpha_i \leq d_{i,R} \tag{4.2}$$

Since the environment is unstructured, the boundaries are absent. Therefore, we establish the boundary tube to prevent the agent from colliding with obstacles. Details are provided in Subsection 4.4 The 2D position vector of the path is given by $\mathbf{s}_i \triangleq [x_i, y_i]^\top$. Along the x dimension, the path can be represented using cubic spline as follows:

$$\begin{aligned} x_i(t_i) &= a_i + b_i t_i + c_i t_i^2 + d_i t_i^3 \\ x_i'(t_i) &= b_i + 2c_i t_i + 3d_i t_i^2 \\ x_i''(t_i) &= 2c_i + 6d_i t_i \\ t_i(s) &= \frac{s - s_{i0}}{\Delta s_i} \end{aligned} \tag{4.3}$$

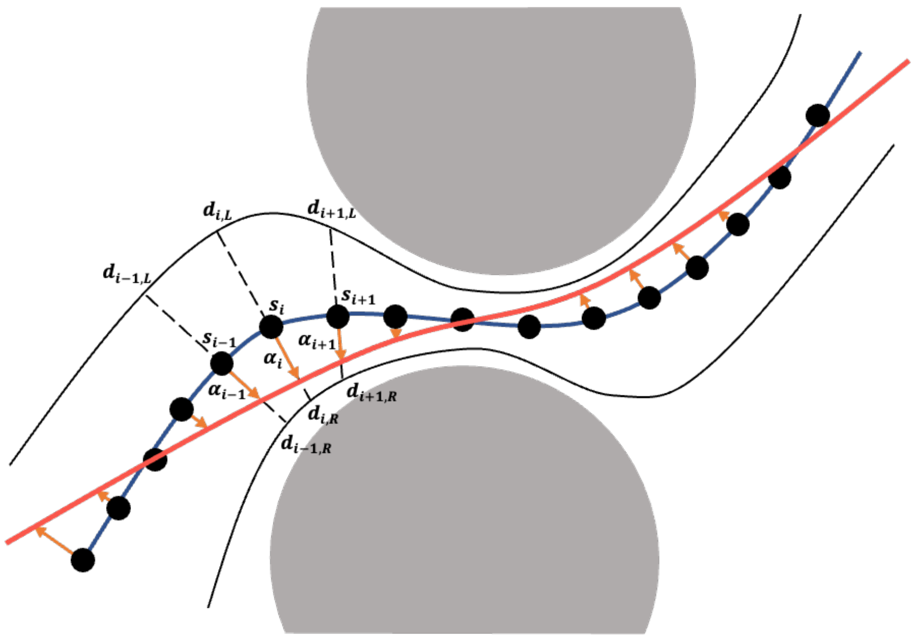


Fig. 2: The black points represent the reference path with boundaries defined by $d_{i,L}$ and $d_{i,R}$. The parameter α indicates the deviation in the normal direction of the path, and the red path represents the new path.

where t_i is a normalized curvilinear parameter of each segment, ranging from 0 to 1. So $x_i(0)$ denotes the start of segment i, and $x_i(1)$ denotes the end of segment i. The spline for y_i follows similarly to Equation (4.3). The a_i, b_i, c_i, and d_i are spline coefficients that define the shape of each cubic segment. These coefficients are derived from a system of linear equations, which enforce constraints such as continuity of the path and its derivatives, as well as specific boundary conditions. For a detailed derivation, refer to Section A.

4.3. Minimum Curvature Path with Energy Penalty. The objective function aims to minimize the sum of squared curvatures and the penalty term for the path's energy, as follows:

$$\begin{aligned} &\text{minimize} \sum_{i=1}^{N} \kappa_i^2 + \gamma E_i \quad \text{where} \quad E_i = \int_0^1 \left\| \frac{d\mathbf{s}_i}{dt} \right\|^2 dt \\ &\text{s.t.} \quad \kappa_i \leq \kappa_{\text{bound}}, \quad \alpha_i \in [d_{i,L}, d_{i,R}], \quad \forall i \in \{1, 2, \dots, N\} \end{aligned} \tag{4.4}$$

Here, κ_i represents the curvature at point i and κ_{bound} is the maximum allowable curvature, where there are N points in total.

Unlike racing scenarios where the path boundaries are fixed (e.g., [12, 15]), in our case, the boundary tube is dynamic and can be adjusted during each iteration. This flexibility helps to make a smooth path, but sometimes it makes the path unnecessarily long, especially in areas with no obstacles nearby. We introduce an additional term, E_i, with a weighting factor, γ. Here, E_i can be interpreted as the energy of the path, which achieves a similar result to minimizing the path length. However, instead of using the path length, represented in cubic splines as $L_i = \int_0^1 \left\| \frac{d\mathbf{s}_i}{dt} \right\| dt$, we use the energy of the path. This is because L_i results in a non-quadratic objective function. Using the squared velocity norm instead, we transform the objective into a quadratic form, allowing it to be solved as a QP problem.

The QP problem is formulated in matrix form, which follows the objective described in Equation (4.4). The QP formulation can be expressed as follows:

$$\text{minimize} \quad \boldsymbol{\alpha}^\top (H_\kappa + H_E)\boldsymbol{\alpha} + (f_\kappa + f_E)^\top \boldsymbol{\alpha} \quad \text{s.t.} \quad E\boldsymbol{\alpha} \leq \mathbf{k} \tag{4.5}$$

The detailed derivation of Equation (4.5) is provided in Section B. When deriving the formulation in matrix form, it is challenging to establish a general QP form because the first derivatives, x_i' and y_i', are variables. However, by adopting the constant-derivative approximation inspired by Heilmeier et al. [12], this can be addressed by assuming that the path heading deviates only slightly from the reference path (or the previous path). Under the assumption, the first derivatives can be approximated as constants. Then, the problem is solved iteratively, updating the linearized variables in each step to minimize the linearization error. As a result, the solution converges to an optimized path that satisfies both curvature and energy efficiency requirements.

Remark 4.1 (Comparison with Minimum-Time Path). The minimum time path in racing is a compromise between the curvature-optimal path and the shortest path [4]. Using the path's energy term, this approach still achieves a similar effect to minimizing the path length, resulting in a path that is closer to the time-optimal solution.

4.4. Improving Safety and Stability of the Proposed Trajectory Optimizer.

The proposed trajectory optimizer employs dynamic boundary tubes to maintain safe margins and adapts curvature bounds iteratively to enhance stability. These strategies ensure that the optimization remains stable and generates safe and feasible trajectories, even in unstructured environments.

4.4.1. Dynamic Boundary Tube Generation.

A boundary tube is necessary to form boundary conditions in the QP optimization. At each path point, the distance to the nearest obstacle in the normal direction, denoted as d_i, is computed. The distance is constrained to lie within the range $[0, d_{\max}]$, where $d_{\max}$ is the maximum allowable tube size. To ensure smooth transitions along the path, the rate of change of this distance, $\dot{d}_i$, is further constrained by a maximum rate, $\dot{d}_{\max}$. This constraint mitigates abrupt variations in the boundary tube, thereby enhancing the stability of the optimization problem. Moreover, as the optimization progresses, both the path and the boundary tube dynamically adapt to nearby obstacles, ensuring safe and feasible trajectory generation in unstructured environments.

However, a notable problem during optimization is that boundary tubes of different path segments may intersect, as illustrated in Figure 3a. Such intersections can cause optimization failures and result in unsafe paths, as the optimized trajectory may become tangled or invalid. This problem typically occur in two scenarios: (1) in sharp corner regions, where the geometric configuration causes the boundary to cross, and (2) due to linearization errors in the QP formulation, or numerical error in the optimization (e.g., numerical precision error of a large matrix or ill-conditioned Hessian matrix).

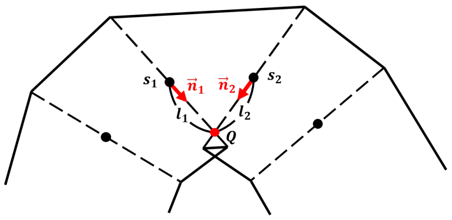
To solve this problem, the tube correction is applied by trimming out the boundary intersection. For each consecutive path point, given $\mathbf{s}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}^\top$, we check for intersections when the boundaries are expanded in normal direction at each point.

If an intersection occurs, the boundary distances are adjusted by reducing them as follows:

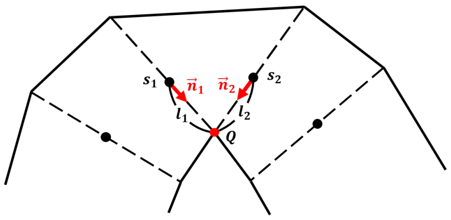
$$l_1 = \frac{n_{2x}(y_2 - y_1) - n_{2y}(x_2 - x_1)}{\Delta} \tag{4.6a}$$

$$l_2 = \frac{n_{1x}(y_2 - y_1) - n_{1y}(x_2 - x_1)}{\Delta} \tag{4.6b}$$

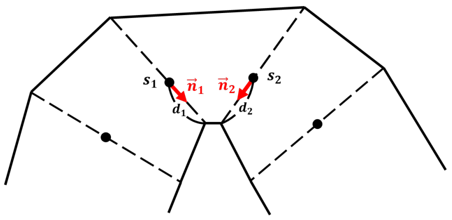
where $\Delta = -n_{1x}n_{2y} + n_{2x}n_{1y}$. Section C shows the detailed derivation of Equation (4.6). Note that if $\Delta = 0$, the normal vectors are parallel, and no intersection can occurs. To further prevent crossings, each boundary distance is reduced by a small value ϵ. The updated boundary distances are then computed as $d_i = l_i - \epsilon$ for $i = 1, 2$.



(a) Boundary tube crossing



(b) Cutting and reassembling the tube



(c) Additional cutting

Fig. 3: Illustration of boundary correction methods

4.4.2. Decaying Curvature Bound. In solving the QP problem, the given curvature constraint may impose an infeasible constraint that the initial path cannot satisfy, since the initial path is generated from such A* or RRT family that cannot consider curvatures. Consequently, the iterative QP optimization process may fail immediately in the first iteration. To address this, the iterative optimization starts with a relaxed (larger) curvature bound, gradually tightening it to meet our desired constraint. The curvature bound at each iteration j is defined as:

$$\kappa_{\text{bound}} = (\kappa_{\max} - \kappa_{\min})e^{-\sigma j} + \kappa_{\min} \tag{4.7}$$

where $\kappa_{\max}$ is a curvature bound at the first iteration, $\kappa_{\min}$ is the desired curvature bound, and σ is a decaying rate of the curvature bound. This approach helps to find a feasible solution by allowing a sufficiently large solution space in the initial iterations, preventing failures in the high-curvature path section. Even if the optimization cannot fully meet the desired curvature constraint, the proposed approach ensures that the resulting path approximates the desired constraint as closely as possible.

4.5. Improving Efficiency of the Proposed Trajectory Optimizer. We make several additional improvements to our proposed optimization method in order to improve computation efficiency.

4.5.1. Adaptive Spacing. We use two path spacing methods. One is uniform path spacing, which divides the path into equal intervals. It is the simplest method to implement and allows for control inputs to be applied consistently. So it is used for providing control information after the optimization is complete.

However, the spacing distance affects the ability to detect surrounding obstacles, e.g, a large spacing might fail to detect obstacles that are in between the space. Hence, assigning the same spacing interval, regardless of the presence of nearby obstacles, is inefficient, and unnecessarily increases the optimization complexity when there are not many obstacles around. To resolve this, we proposed and implemented adaptive path spacing, which adjusts the spacing according to the distance to the closest obstacles, ranging from $s_{\text{adaptive, min}}$ to $s_{\text{adaptive, max}}$. As a result, the path points become denser near obstacles and sparser in obstacle-free areas, effectively reducing the number of points and the computational load during optimization.

4.5.2. Stopping Criteria. Common stopping criteria for any iterative algorithm include terminating the process after a fixed number of iterations or when the change in optimization metrics falls within a small threshold. However, such criteria are unsuitable for unstructured environments, where performance metrics vary depending on the conditions. A more efficient stopping criterion can be established by considering the expected travel time, calculated using a velocity profile from the optimized path. If the new travel time exceeds that of the previous iteration, the process is terminated and the previous path is used. Otherwise, optimization continues until the maximum number of iterations is reached.

4.6. Velocity Profile Generation. To ensure the agent navigates smoothly and safely, a velocity profile is generated to regulate its speed at each point. The profile defines a sequence of speed values along the path, constraining both the longitudinal and lateral accelerations of the agent. The constraints on the accelerations are defined as $|a_{\text{long}}| \leq a_{\text{max, long}}$ and $|a_{\text{lat}}| \leq a_{\text{max, lat}}$. Here, a_{long} and a_{lat} represent the longitudinal and lateral accelerations, respectively, with their maximum allowable values denoted by $a_{\text{max, long}}$ and $a_{\text{max, lat}}$. The constraints on the longitudinal and lateral velocities are now given as follows:

$$v_{\text{long}} = \sqrt{v_{\text{long, prev}}^2 + 2\, a_{\text{max, long}}\, \Delta s} \tag{4.8a}$$

$$v_{\text{lat}} = \sqrt{a_{\text{max, lat}}/\kappa} = \sqrt{R\, a_{\text{max, lat}}} \tag{4.8b}$$

where κ denotes the curvature, R the turning radius, and Δs the distance over which acceleration is applied. This formulation ensures the agent's speed is finely adjusted based on the path's curvature and other dynamic requirements, enhancing control precision and safety.

Other constraints can be considered if necessary. For example, the maximum permissible velocity is also taken into account. The final velocity of the agent is determined by taking the minimum of the longitudinal and lateral velocity constraints and the maximum velocity constraint:

$$v_{\text{final}} = \min\{v_{\text{long}}, v_{\text{lat}}, v_{\text{max}}\} \tag{4.9}$$

5. Numerical Experiment. We demonstrate our proposed approach using a 4-wheel mobile robot. Assuming simplified lateral dynamics, the vehicle's motion can be described by the bicycle model with the following kinematic equations:

$$\begin{aligned} \dot{X} &= V\cos(\psi) \\ \dot{Y} &= V\sin(\psi) \\ \dot{\psi} &= \frac{V}{L}\tan(\delta) \end{aligned} \tag{5.1}$$

Here, V, ψ represent the velocity and heading angle of vehicle, respectively, δ is the steering angle, and L represents the wheelbase.

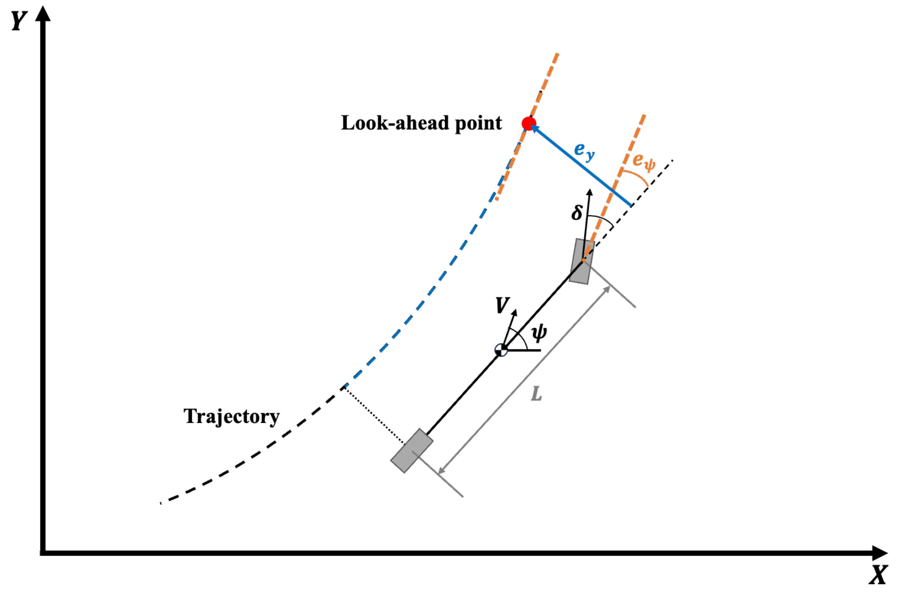


Fig. 4: Schematic of the bicycle model and path tracking errors.

5.1. Trajectory Tracking Controller. To design the path-following controller, we separate the lateral control and longitudinal control. A standard Linear Parameter Varying (LPV) model [28] is used as a control model, and we utilize a Linear Quadratic Regulator (LQR) for the lateral control. The longitudinal controller uses PID control to follow the velocity profile obtained from Subsection 4.6. The LPV system equation is given as follows:

$$\begin{bmatrix} \dot{e}_{\mathrm{y}} \\ \dot{e}_{\psi} \end{bmatrix} = \begin{bmatrix} 0 & V \\ 0 & 0 \end{bmatrix} \begin{bmatrix} e_{\mathrm{y}} \\ e_{\psi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{V}{L} \end{bmatrix} u \tag{5.2}$$

Here, e_y represents the lateral error, defined as the shortest distance between agent's axis and the reference path at the look-ahead point. e_ψ denotes the heading error, which is the angular difference between agent's heading and the direction of the reference at the look-ahead point. V is the agent's velocity, L is the wheelbase of the vehicle, and u represents the control input which is steering angle.

The LQR gain is obtained by minimizing the following cost function:

$$J = \int_0^\infty \left(x^\top Qx + u^\top Ru\right) dt \tag{5.3}$$

Since the model is an LPV system, the control gain, K, depends on the velocity, V. Since solving the Riccati equation in real-time is computationally inefficient, we employed gain scheduling by precomputing the solutions to the Riccati equation for a range of velocities and used these precomputed gains during runtime.

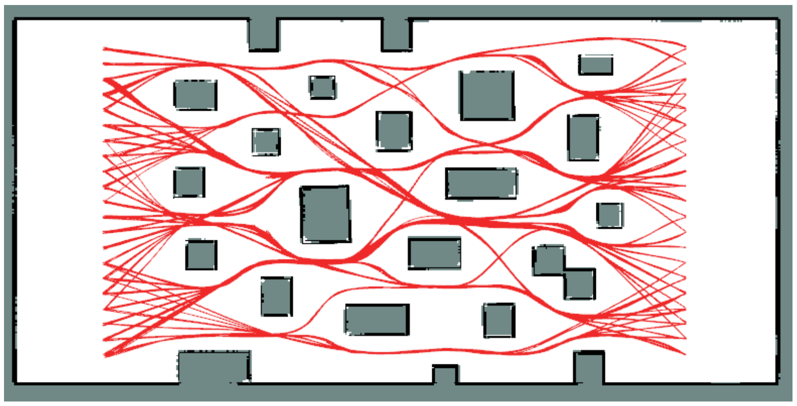


Fig. 5: Environment used in the experiment. The lines shows all possible paths.

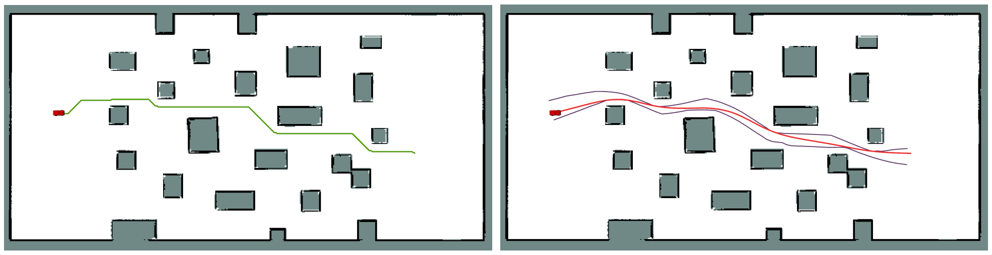
5.2. Simulation of the Proposed Trajectory Optimization Algorithm. The simulation experiments were conducted on a computer equipped with an Intel Core i7-14700k processor running Ubuntu 18.04. To evaluate the proposed trajectory optimization method, we used the open-source autonomous vehicle simulator F1Tenth [1]. A global cost map is generated using SLAM, namely Google's Cartographer [13]. To optimize the QP, we employed the OSQP solver [35]. Figure 5 illustrates the environment used in the experiment, and shows all 441 possible paths from all 21 possible starting locations to all 21 possible goals. The map measures 25m $\times$ 12m map. The experiment was carried out under two inflation layer configurations in the ROS1 *navcore* framework: a thick case (*inflation_radius*: 0.5, *cost_scaling_factor*: 1.0) and a thin case (*inflation_radius*: 0.35, *cost_scaling_factor*: 1.0). The key simulation parameters for this experiment are summarized in Table 1.

Additionally, an initial path was generated using the A* algorithm. Before optimizing the path, we pre-process the path obtained from A* by applying the path smoothing technique [3] to eliminate unnecessary points and make the path short. This pre-processed path served as the input for the proposed optimization method. Figure 6 shows the initial path generated from A* and the final path obtained from the optimization with boundary tube.

Table 1: Simulation parameters used in the experiment.

Parameter Name	**Value**	**Unit**	**Parameter Name**	**Value**	**Unit**
$v_{\max}$	5.0	m/s	s_{uniform}	0.25	m
$a_{\text{max, long}}$	1.0	m/s^2	$s_{\text{adaptive, min}}$	0.25	m
$a_{\text{max, lat}}$	2.0	m/s^2	$s_{\text{adaptive, max}}$	0.50	m
$\kappa_{\min}$	0.5	rad/m	$d_{\max}$	0.75	m
$\kappa_{\max}$	1.5	rad/m	$\dot{d}_{\max}$	0.05	m/s
σ	0.6	-	M	10	-
γ	1×10^{-8}	-			

* All parameters are in SI units where applicable. M indicates the maximum number of iterations in the iterative QP process.



(a) (b)

Fig. 6: (a) Initial path generation using A*. (b) Final optimized path with boundary tube.

5.3. Results. We compared the performance of three trajectory optimization methods: QP, QP with energy penalty (QP+EP), and B-spline. Here, B-spline is applied to the path obtained from A*, incorporating the path smoothing technique described in Subsection 5.2. The evaluation includes computation time of trajectory optimization, optimization success rate of QP-based methods, travel time, and the success rate of path tracking, which measures whether the robot successfully reaches the goal.

5.4. Computation time and Optimization Success Rates Analysis. Computation time indicates the calculation time of each smoothing algorithm. Optimization success rate evaluates the success of QP optimization, requiring at least 3 iterations to ensure the path is sufficiently well-optimized. The computation times are only measured when optimization succeeds.

B-spline showed the lowest computation time among the three methods, as it does not involve any optimization process. Thus, the optimization success rate is not evaluated for this method. For QP and QP+EP, the computation times and optimization success rates were nearly identical. Despite the inclusion of the energy penalty term in QP+EP, it did not result in a noticeable increase in computation time.

5.5. Travel Time and Path Tracking Success Rate Analysis. Travel time represents the time for an agent to navigate from start to goal. Path tracking success rate measures the rate that the agent successfully reached the goal without collisions.

Table 2: Computation Time and Optimization Success Rate for Thick and Thin Inflation Layers

Method	**Computation Time (s)**						**Optimization**
Name	**Min**	**Q1**	**Q2**	**Q3**	**Max**	**Mean**	**Success Rate (%)**
Thick Inflation Layer							
QP	0.82	2.30	2.84	3.12	3.87	2.67	95.46
QP+EP	0.79	2.29	2.88	3.16	3.88	2.69	95.01
B-spline	0.03	0.05	0.06	0.07	0.11	0.06	N/A
Thin Inflation Layer							
QP	0.88	1.43	1.95	2.44	3.30	1.95	94.56
QP+EP	0.87	1.43	1.94	2.42	3.16	1.94	95.46
B-spline	0.00	0.06	0.07	0.08	0.11	0.07	N/A

* All values are rounded to the second decimal place. Computation times are measured in seconds. Optimization success rate represents the percentage of runs with at least three iterations. The computation time 0.00 indicates a value less than 0.01.

Table 3: Travel Time and Path Tracking Success Rate for Thick and Thin Inflation Layers

Method	**Travel Time (s)**						**Path Tracking**
Name	**Min**	**Q1**	**Q2**	**Q3**	**Max**	**Mean**	**Success Rate (%)**
Thick Inflation Layer							
QP	8.26	9.01	9.30	9.65	15.02	9.49	100.00
QP+EP	8.03	8.97	9.27	9.63	14.90	9.45	100.00
B-spline	8.12	9.92	10.49	11.04	12.33	10.42	100.00
Thin Inflation Layer							
QP	7.60	8.34	8.66	8.95	13.49	8.74	96.15
QP+EP	7.59	8.16	8.56	8.86	13.53	8.66	96.83
B-spline	7.62	8.71	9.17	9.55	10.83	9.14	35.15

* All values are rounded to the second decimal place. Travel times are measured in seconds.

The travel times are recorded only in cases where path tracking was successful.

As shown in Table 3, thinner inflation layers reduce travel time by allowing the agent to navigate closer to corners, resulting in shorter paths with lower curvature. However, thinner layer increase collision risks due to insufficient sufficient boundaries, leading to lower path tracking success rates. This limitation is particularly apparent in B-spline case, which has the lowest path tracking success rate among the evaluated methods, since its convex hull property causes paths to pass dangerously closer to obstacles, increasing collision risks. In contrast, QP-based methods mitigate this problem by incorporating boundary tube constraints, which provide safety margins. This approach ensures higher success rates even in thin inflation layer cases. Nevertheless, the success rate decreases in thin layers, compared to in thick layers. In such cases, QP-based methods can further utilize boundary tubes to create tighter safety margins by reducing the tube size.

Additionally, QP+EP achieved the shortest travel time, followed by QP and B-spline. This result aligns with the remark statement discussed in Subsection 4.3. To navigate safely, B-splines must reduce their smoothness, which leads to higher curvature and more restrictive lateral acceleration limits. Conversely, QP-based methods minimize path curvature within the safety boundary tube, reducing the impact of lateral acceleration limits. As a result, QP-based methods achieve higher velocity profiles, ultimately reducing travel time.

6. Conclusions.

We proposed a trajectory optimization algorithm that extends curvature-based optimization method, traditionally used in structured settings like racing tracks, to unstructured environments with complex obstacle distributions. By incorporating a boundary tube, our method ensures safe and efficient navigation in these challenging environments. In addition, we introduced a curvature-energy optimization strategy that balances smoothness with path's energy. This approach leads to near time-optimal trajectories, achieving a compromise between minimizing curvature and reducing distance. Also, the method simplifies the complex optimization problem with dynamics into a QP formulation, enabling efficient computation.

To generate a safe and stable path, a method that corrects the ill-conditioned boundary tube is introduced. Decaying curvature bound is also introduced to find a feasible solution to prevent failures of the optimization and obtain an optimal solution in the best effort. Additionally, to improve efficiency, we introduced adaptive path spacing to reduce problem dimensionality and a stopping rule to ensure timely termination of the optimization process. These improvements collectively enhance the algorithm's stability, safety, and computational efficiency, making it suitable for both well-structured and unstructured environments.

The proposed QP-based algorithm demonstrated better performance compared to the B-spline in terms of path tracking success rate and travel time, while the B-spline achieved the shortest computation time due to the absence of an optimization step. The experiments showed a trade-off where thinner inflation layers resulted in shorter travel times but reduced path tracking success rates. Summarizing the experimental results, the QP and QP+EP (QP with energy penalty) showed no significant difference in computation time, while the B-spline demonstrated the fastest computation time due to the absence of an optimization step. Safety problem occurred in the B-spline due to its convex hull property. To address this problem, the smoothness needs to be reduced, which results in higher curvature paths and longer travel times. In contrast, the QP-based methods effectively utilized boundary tube to ensure safe navigation in the complex environments. This approach allowed the QP-based methods to maintain lower curvature safely, achieve higher velocity profiles, and ultimately record shorter travel time. Lastly, QP+EP achieved the shortest travel time among the methods by compromising path smoothness and distance.

Despite these improvements, some limitations remain that present opportunities for further improvement. No matter how well an optimal curvature path is obtained, the final path remains dependent on the initial path. Thus, there could still be a better path. Therefore, identifying the optimal homotopy for minimizing curvature or time can be a future work to improve overall efficiency of the path.

The penalty factor, γ is constant along the entire path. However, performance could be further improved by variable penalty factor γ_i on each path's point. For instance, γ could be increase in low-curvature regions to prioritize minimizing distance, which being reduced in high-curvature regions to emphasize curvature minimization. Dynamically predicting appropriate γ_i values could improve the approximation of the optimal time trajectory.

Finally, while the iterative QP-based algorithm demonstrates strong performance in unstructured environments and generates smooth and feasible paths, it is not yet suitable for real-time or online path optimization due to its computational complexity. This limitation is particularly significant in dynamic environments, where paths must be adjusted rapidly to avoid moving obstacles. In such cases, maintaining the optimized path's properties is crucial for ensuring path's consistency and reliability. Future research will focus on improving the algorithm's efficiency and developing strategies to preserve path quality in real-time scenarios.

Appendix A. Derivation of Cubic Spline Formulation.

As described in Equation (4.3), the path is represented as cubic spline. The entire path satisfies C^0, C^1, and C^2 continuity, ensuring smoothness in both position and derivatives. This is achieved by matching the function and derivative values at the boundaries of the spline segments:

$$\begin{aligned} x_i(t=1) &= x_{i+1}(t=0) &&\Leftrightarrow && a_i + b_i + c_i + d_i = a_{i+1} \\ x_i'(t=1) &= x_{i+1}'(t=0) &&\Leftrightarrow && b_i + 2c_i + 3d_i = b_{i+1} \\ x_i''(t=1) &= x_{i+1}''(t=0) &&\Leftrightarrow && 2c_i + 6d_i = 2c_{i+1} \end{aligned} \tag{A.1}$$

To fully define the spline, boundary conditions must be applied at the start and end points. In this case, a natural boundary condition is used, which sets the second derivatives of the spline at the endpoints to zero. Thus, the polynomial constraints are derived as follows:

$$\begin{aligned} x_1''(t=0) &= 0 &&\Leftrightarrow && 2c_1 = 0 \\ x_N''(t=1) &= 0 &&\Leftrightarrow && 2c_N + 6d_N = 0 \end{aligned} \tag{A.2}$$

The above conditions for all N segments can be compactly represented as a global linear system $A\mathbf{z} = \mathbf{b}$, as follows:

$$\left[\begin{array}{cccccccccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & & & & & & & \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & & & & & & & & \\ 0 & 1 & 2 & 3 & 0 & -1 & 0 & 0 & & & & & & & & \\ 0 & 0 & 2 & 6 & 0 & 0 & -2 & 0 & & & & & & & & \\ & & & & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & & & \\ & & & & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & & & & \\ & & & & 0 & 1 & 2 & 3 & 0 & -1 & 0 & 0 & & & & \\ & & & & 0 & 0 & 2 & 6 & 0 & 0 & -2 & 0 & & & & \\ & & & & & & & & \ddots & \ddots & \ddots & \ddots & & & & \\ & & & & & & & & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & & & & & & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ & & & & & & & & 0 & 1 & 2 & 3 & 0 & -1 & 0 & 0 \\ & & & & & & & & 0 & 0 & 2 & 6 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & & & & & & & & & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & & & & & & & & & 1 & 1 & 1 & 1 \\ 0 & 0 & 2 & 0 & & & & & & & & & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & & & & & & & & & 0 & 0 & 2 & 6 \end{array}\right] \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \\ \vdots \\ a_{N-1} \\ b_{N-1} \\ c_{N-1} \\ d_{N-1} \\ a_N \\ b_N \\ c_N \\ d_N \end{bmatrix} = \begin{bmatrix} x_1(t=0) \\ x_1(t=1) \\ 0 \\ 0 \\ x_2(t=0) \\ x_2(t=1) \\ 0 \\ 0 \\ \vdots \\ x_{N-1}(t=0) \\ x_{N-1}(t=1) \\ 0 \\ 0 \\ x_N(t=0) \\ x_N(t=1) \\ x_N''(t=0) \\ x_N''(t=1) \end{bmatrix} \tag{A.3}$$

To extract the coefficients a, b, c, d for each spline segment, the global linear system $A\mathbf{z} = \mathbf{b}$ is solved. After solving for $\mathbf{z}$, the coefficients a_i, b_i, c_i, d_i for each segment i are directly obtained from the corresponding positions in the solution vector. The matrices A_c and $A_{b,c,d}$ are used to extract the coefficients c and b, c, d, respectively, as will be described in Section B.

Appendix B. Derivation of the Proposed QP in a Matrix Form. In this Appendix, we derive the QP problem in the matrix form, which is shown in Equation (4.5). First, the matrix form of curvature objective is derived in Subsection B.1. Second, the matrix form of path's energy objective is derived in Subsection B.2. Third, the inequality constraint is derived in Subsection B.3. Finally, the combined QP problem is derived in Subsection B.4.

B.1. Derivation of QP for Minimum Curvature. The curvature and its square at each point on a path can be expressed as:

$$\begin{aligned}\kappa_i &= \frac{x_i'y_i'' - y_i'x_i''}{(x_i'^2 + y_i'^2)^{3/2}} \\ \kappa_i^2 &= \frac{x_i'y_i'' - 2x_i'x_i''y_i'y_i'' + y_i'^2x_i''^2}{(x_i'^2 + y_i'^2)^3}\end{aligned} \tag{B.1}$$

Here, x_i' and y_i' represent the first derivatives, and x_i'', y_i'' are the second derivatives of the path coordinates. The objective is to minimize the total squared curvatures at each point along the path:

$$\min \sum_{i=1}^{N} \kappa_i^2 \quad \text{subject to} \quad \alpha_i \in [d_{i,L}, d_{i,R}] \tag{B.2}$$

Defining vectors for the second derivatives:

$$\mathbf{x}'' = [x_1'', \dots, x_N'']^\top, \quad \mathbf{y}'' = [y_1'', \dots, y_N'']^\top \tag{B.3}$$

We form the quadratic programming problem in matrix form, as follows:

$$\begin{aligned}&\min \mathbf{x}''^\top P_{xx}\mathbf{x}'' + \mathbf{y}''^\top P_{xy}\mathbf{x}'' + \mathbf{y}''^\top P_{yy}\mathbf{y}'' \\ &\text{s.t.} \quad \alpha_i \in [\alpha_{i,\min}, \alpha_{i,\max}] \quad \forall i \in \{1, \dots, N\}\end{aligned} \tag{B.4}$$

Note: We assume that the path heading only changes slightly, compared to reference path (previous path). Thus, P_{xx}, P_{xy}, and P_{yy} are constant.

$$P_{xx} = \operatorname{diag}\left(\frac{y_1'^2}{(x_1'^2 + y_1'^2)^3}, \frac{y_2'^2}{(x_2'^2 + y_2'^2)^3}, \dots, \frac{y_N'^2}{(x_N'^2 + y_N'^2)^3}\right) \tag{B.5a}$$

$$P_{xy} = \operatorname{diag}\left(\frac{-2x_1'y_1'}{(x_1'^2 + y_1'^2)^3}, \frac{-2x_2'y_2'}{(x_2'^2 + y_2'^2)^3}, \dots, \frac{-2x_N'y_N'}{(x_N'^2 + y_N'^2)^3}\right) \tag{B.5b}$$

$$P_{yy} = \operatorname{diag}\left(\frac{x_1'^2}{(x_1'^2 + y_1'^2)^3}, \frac{x_2'^2}{(x_2'^2 + y_2'^2)^3}, \dots, \frac{x_N'^2}{(x_N'^2 + y_N'^2)^3}\right) \tag{B.5c}$$

Since the second derivative of the spline at $t = 0$ is given by $\mathbf{x}''(t = 0) = 2\mathbf{c}$, the second derivative of the spline, $\mathbf{x}''$, can be expressed in terms of the optimization variable $\boldsymbol{\alpha}$:

$$\mathbf{x}'' = 2A_cA^{-1}(\mathbf{p}_x + N_x\boldsymbol{\alpha}) = T_c\mathbf{p}_x + T_{n,x}\boldsymbol{\alpha} \tag{B.6}$$

Here, A_c is a extraction matrix of c, $T_c = 2A_cA^{-1}$ and $T_{n,x} = 2A_cA^{-1}N_x$ are transformation matrices derived from the cubic spline matrix A, and $\mathbf{p}_x$ is the position vector in the x-dimension. Similarly, the second derivative in the y-dimension, $\mathbf{y}''$, and the first derivatives, $\mathbf{x}'$ and $\mathbf{y}'$, can be computed using analogous formulations.

Inserting Equation (B.6) and the corresponding version for y in Equation (B.4), we finally obtain:

$$\begin{aligned} &\min \boldsymbol{\alpha}^\top (H_x + H_{xy} + H_y)\boldsymbol{\alpha} + (\mathbf{f}_x + \mathbf{f}_{xy} + \mathbf{f}_y)^\top \boldsymbol{\alpha} + \text{const.} \\ &\quad \text{s.t. } \alpha_i \in [\alpha_{i,\min}, \alpha_{i,\max}] \quad \forall i \in \{1, \dots, N\} \end{aligned} \tag{B.7}$$

where

$$\begin{aligned} H_x &= T_{n,x}^\top P_{xx} T_{n,x}, \\ H_{xy} &= T_{n,y}^\top P_{xy} T_{n,x}, \\ H_y &= T_{n,y}^\top P_{yy} T_{n,y}, \\ \mathbf{f}_x &= 2T_{n,x}^\top P_{xx} T_c \mathbf{q}_x, \\ \mathbf{f}_{xy} &= T_{n,y}^\top P_{xy} T_c \mathbf{q}_x + T_{n,x}^\top P_{xy} T_c \mathbf{q}_y, \\ \mathbf{f}_y &= 2T_{n,y}^\top P_{yy} T_c \mathbf{q}_y, \\ \text{const} &= \mathbf{q}_x^\top T_c^\top P_{xx} T_c \mathbf{q}_x + \mathbf{q}_y^\top T_c^\top P_{xy} T_c \mathbf{q}_x + \mathbf{q}_y^\top T_c^\top P_{yy} T_c \mathbf{q}_y. \end{aligned}$$

B.2. Derivation of QP for Energy Penalty. The cubic spline path representations along x and y and its derivation are as follows:

$$\begin{aligned} x_i(t) &= a_i + b_i t + c_i t^2 + d_i t^3 \\ y_i(t) &= e_i + f_i t + g_i t^2 + h_i t^3 \\ \dot{x}_i(t) &= b_i + 2c_i t + 3d_i t^2 \\ \dot{y}_i(t) &= f_i + 2g_i t + 3h_i t^2 \end{aligned} \tag{B.8}$$

The energy penalty term in Equation (4.4) is defined as:

$$E_i = \int_0^1 \|\frac{d\mathbf{s}_i}{dt}\|^2 \, dt = \int_0^1 \left(\dot{x}_i^2 + \dot{y}_i^2\right)(t) \, dt \tag{B.9}$$

Then separate the energy term into x-component and y-component, denoted as $E_{x,i} = \int_0^1 \dot{x}_i^2(t)\, dt$ and $E_{y,i} = \int_0^1 \dot{y}_i^2(t)\, dt$, respectively:

$$\begin{aligned} E_{x,i} &= b_i^2 + 2b_i c_i + 2b_i d_i + \frac{4}{3}c_i^2 + 3c_i d_i + \frac{9}{5}d_i^2 \\ E_{y,i} &= f_i^2 + 2f_i g_i + 2f_i h_i + \frac{4}{3}g_i^2 + 3g_i h_i + \frac{9}{5}h_i^2 \end{aligned} \tag{B.10}$$

Let us define $Q_{L,i} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 4/3 & 3/2 \\ 1 & 3/2 & 9/5 \end{bmatrix} \succ 0$. Then we can define E which is the energy term along the entire path as a matrix form as follows:

$$\begin{aligned} E &= \sum_{i=1}^{N} E_i = \mathbf{q}_x^\top Q_E \mathbf{q}_x + \mathbf{q}_y^\top Q_E \mathbf{q}_y \\ \text{where} \quad Q_E &= diag(Q_{L,1}, Q_{L,2}, \cdots, Q_{L,N}) \\ \mathbf{q}_x &= [b_1, c_1, d_1, b_2, c_2, d_2, \cdots, b_N, c_N, d_N]^\top \\ \mathbf{q}_y &= [f_1, g_1, h_1, f_2, g_2, h_2, \cdots, f_N, g_N, h_N]^\top \end{aligned} \tag{B.11}$$

The vectors $\mathbf{q}_x$ and $\mathbf{q}_y$ are expressed in terms of the position vector $\mathbf{p}_x$, $\mathbf{p}_y$ and the optimization variable $\boldsymbol{\alpha}$:

$$\begin{aligned} \mathbf{q}_x &= A_{b,c,d}A^{-1}(\mathbf{p}_x + N_x\boldsymbol{\alpha}) = T_q\mathbf{p}_x + T_{qnx}\boldsymbol{\alpha} \\ \mathbf{q}_y &= A_{b,c,d}A^{-1}(\mathbf{p}_y + N_y\boldsymbol{\alpha}) = T_q\mathbf{p}_y + T_{qny}\boldsymbol{\alpha} \\ \text{where} \quad T_q &= A_{b,c,d}A^{-1},\ T_{qnx} = A_{b,c,d}A^{-1}N_x,\ \text{and } T_{qny} = A_{b,c,d}A^{-1}N_y \end{aligned} \tag{B.12}$$

Here, $A_{b,c,d}$ is a extraction matrix of b, c, and d. Finally, substituting these expressions into the energy term gives:

$$\begin{aligned} E = \boldsymbol{\alpha}^\top \left(T_{qnx}^\top Q_E T_{qnx} + T_{qny}^\top Q_E T_{qny}\right)\boldsymbol{\alpha} + \left(2T_{qnx}^\top Q_E T_q\mathbf{p}_x + 2T_{qny}^\top Q_E T_q\mathbf{p}_y\right)^\top \boldsymbol{\alpha} \\ + \mathbf{p}_x^\top T_q^\top Q_E T_q \mathbf{p}_x + \mathbf{p}_y^\top T_q^\top Q_E T_q \mathbf{p}_y \end{aligned} \tag{B.13}$$

The constant term in the energy calculation does not affect the optimization and can be ignored. Thus, the energy penalty term simplifies to:

$$\begin{aligned} E &= \boldsymbol{\alpha}^\top H_E \boldsymbol{\alpha} + f_E^\top \boldsymbol{\alpha} \\ \text{where} \quad H_E &= T_{qnx}^\top Q_E T_{qnx} + T_{qny}^\top Q_E T_{qny} \\ f_E &= 2T_{qnx}^\top Q_E T_q \mathbf{p}_x + 2T_{qny}^\top Q_E T_q \mathbf{p}_y \end{aligned} \tag{B.14}$$

B.3. Derivation of Constraints. In the optimization problem defined in Equation (4.4), there are two constraints - curvature constraint and boundary constraint. The curvature constraints can be transformed into a linear inequality form to ensure that the path's curvature does not exceed certain limits.

$$|\boldsymbol{\kappa}_{\text{ref}} + \boldsymbol{\kappa}_{\text{var}}| \leq \boldsymbol{\kappa}_{\text{bound}} \tag{B.15}$$

Here, the total curvature is divided into two components: a static part, $\boldsymbol{\kappa}_{\text{ref}}$, derived from the reference line, and a variable part, $\boldsymbol{\kappa}_{\text{var}}$, resulting from shifts along the normal vector. Given the definition of curvature Equation (B.1) and the second derivative obtained in Equation (B.6), the static and variable curvature are expressed as follows:

$$\begin{aligned} \boldsymbol{\kappa}_{ref} &= Q_y T_c \mathbf{p}_y - Q_x T_c \mathbf{p}_x \\ \boldsymbol{\kappa}_{var} &= (Q_y T_{n,y} - Q_x T_{n,x})\boldsymbol{\alpha} = E_k \boldsymbol{\alpha} \end{aligned} \tag{B.16}$$

where

$$\begin{aligned} Q_x &= \text{diag}\left(\frac{y_1'}{(\sqrt{x_1'^2 + y_1'^2})^3}, \frac{y_2'}{(\sqrt{x_2'^2 + y_2'^2})^3}, \ldots, \frac{y_N'}{(\sqrt{x_N'^2 + y_N'^2})^3}\right) \\ Q_y &= \text{diag}\left(\frac{x_1'}{(\sqrt{x_1'^2 + y_1'^2})^3}, \frac{x_2'}{(\sqrt{x_2'^2 + y_2'^2})^3}, \ldots, \frac{x_N'}{(\sqrt{x_N'^2 + y_N'^2})^3}\right) \end{aligned}$$

Additionally, the optimization variable $\boldsymbol{\alpha}$ must satisfy the boundary constraints. Combining these conditions, the resulting linear inequality constraints can be written as:

$$\begin{bmatrix} I_N \\ -I_N \\ E_k \\ -E_k \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{bmatrix} \leq \begin{bmatrix} \mathbf{d}_L \\ \mathbf{d}_R \\ \boldsymbol{\kappa}_{\text{bound}} - \boldsymbol{\kappa}_{\text{ref}} \\ \boldsymbol{\kappa}_{\text{bound}} + \boldsymbol{\kappa}_{\text{ref}} \end{bmatrix} \tag{B.17}$$

where $\mathbf{d}_L = \begin{bmatrix} d_{1,L} & d_{2,L} & \cdots & d_{N,L} \end{bmatrix}^\top$, $\mathbf{d}_R = \begin{bmatrix} d_{1,R} & d_{2,R} & \cdots & d_{N,R} \end{bmatrix}^\top$, and I_N is the identity matrix with size N.

B.4. Combined QP. The objective function we propose is a combination of curvature and energy of path. Since both the curvature and energy objectives are convex and quadratic, we can combine them directly. The weighting factor γ balances the trade-off between minimizing curvature and energy efficiency. Thus the final QP problem is expressed as:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\alpha}^\top (H_\kappa + H_E)\boldsymbol{\alpha} + (f_\kappa + f_E)^\top \boldsymbol{\alpha} \\ \text{s.t.} \quad & E\boldsymbol{\alpha} \leq \mathbf{k} \end{aligned} \tag{B.18}$$

Appendix C. Derivation of Boundary Correction. The boundary correction is applied to both left and right. Here, we derive the correction distances, l_1 and l_2 with respect to the right boundary. We define right-direction vectors in the normal direction of $\mathbf{s}$ as follows:

$$\begin{aligned} \mathbf{r}_1(t) &= \mathbf{s}_1 + l_1\mathbf{n}_1 \\ \mathbf{r}_2(t) &= \mathbf{s}_2 + l_2\mathbf{n}_2 \end{aligned} \tag{C.1}$$

where

$$\mathbf{s}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad \text{and} \quad \mathbf{n}_i = \begin{bmatrix} n_{ix} \\ n_{iy} \end{bmatrix}$$

The intersection of the two right-direction vectors is denoted as a point Q. To find the intersection point, we solve the following equation:

$$\begin{aligned} \mathbf{r}_1(t) = \mathbf{r}_2(t) &\Rightarrow \mathbf{s}_1 + l_1\mathbf{n}_1 = \mathbf{s}_2 + l_2\mathbf{n}_2 \\ &\Rightarrow l_1\mathbf{n}_1 - l_2\mathbf{n}_2 = \mathbf{s}_2 - \mathbf{s}_1 \\ &\Rightarrow \begin{bmatrix} n_{1x} & -n_{2x} \\ n_{1y} & -n_{2y} \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \end{bmatrix} \end{aligned} \tag{C.2}$$

Let us denote the determinant of the normal vector matrix as $\Delta = -n_{1x}n_{2y} + n_{2x}n_{1y}$. If $\Delta = 0$, the normal vectors are parallel, meaning that they are not crossed. Otherwise, the correction distances can be derived by solving the linear equation as follows:

$$l_1 = \frac{n_{2x}(y_2 - y_1) - n_{2y}(x_2 - x_1)}{\Delta} \tag{C.3a}$$

$$l_2 = \frac{n_{1x}(y_2 - y_1) - n_{1y}(x_2 - x_1)}{\Delta} \tag{C.3b}$$

REFERENCES

[1] V. S. Babu and M. Behl, *f1tenth. dev-an open-source ros based f1/10 autonomous racing simulator*, in 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE), IEEE, 2020, pp. 1614–1620.

[2] T. Berglund, A. Brodnik, H. Jonsson, M. Staffanson, and I. Soderkvist, *Planning smooth and obstacle-avoiding b-spline paths for autonomous mining vehicles*, IEEE transactions on automation science and engineering, 7 (2009), pp. 167–172.

[3] A. Botea, M. Müller, and J. Schaeffer, *Near optimal hierarchical path-finding.*, J. Game Dev., 1 (2004), pp. 1–30.

[4] F. Braghin, F. Cheli, S. Melzi, and E. Sabbioni, *Race driver model*, Computers & Structures, 86 (2008), pp. 1503–1516. Structural Optimization.

[5] J.-w. Choi, R. Curry, and G. Elkaim, *Path planning based on bézier curve for autonomous ground vehicles*, in Advances in Electrical and Electronics Engineering-IAENG Special Edition of the World Congress on Engineering and Computer Science 2008, IEEE, 2008, pp. 158–166.

[6] K. Daniel, A. Nash, S. Koenig, and A. Felner, *Theta*: Any-angle path planning on grids*, Journal of Artificial Intelligence Research, 39 (2010), pp. 533–579.

[7] L. E. Dubins, *On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents*, American Journal of mathematics, 79 (1957), pp. 497–516.

[8] P. Foehn, A. Romero, and D. Scaramuzza, *Time-optimal planning for quadrotor waypoint flight*, Science robotics, 6 (2021), p. eabh1221.

[9] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, *Batch informed trees (bit*): Informed asymptotically optimal anytime search*, The International Journal of Robotics Research, 39 (2020), pp. 543–567.

[10] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, *Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic*, in 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2014.

[11] A. K. Guruji, H. Agarwal, and D. Parsediya, *Time-efficient A* algorithm for robot path planning*, Procedia Technology, 23 (2016), pp. 144–149. 3rd International Conference on Innovations in Automation and Mechatronics Engineering 2016, ICIAME 2016 05-06 February, 2016.

[12] A. Heilmeier, A. Wischnewski, L. Hermansdorfer, J. Betz, M. Lienkamp, and B. Lohmann, *Minimum curvature trajectory planning and control for an autonomous race car*, Vehicle System Dynamics, 58 (2020), pp. 1497–1527.

[13] W. Hess, D. Kohler, H. Rapp, and D. Andor, *Real-time loop closure in 2d lidar slam*, in 2016 IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 1271–1278, https://doi.org/10.1109/ICRA.2016.7487258.

[14] L. Janson, E. Schmerling, A. Clark, and M. Pavone, *Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions*, The International Journal of Robotics Research, 34 (2015), pp. 883–921.

[15] C. Jung, A. Finazzi, H. Seong, D. Lee, S. Lee, B. Kim, G. Gang, S. Han, and D. H. Shim, *An autonomous system for head-to-head race: Design, implementation and analysis; team kaist at the indy autonomous challenge*, arXiv preprint arXiv:2303.09463, (2023).

[16] H. Kano and H. Fujioka, *B-spline trajectory planning with curvature constraint*, in 2018 Annual American Control Conference (ACC), IEEE, 2018, pp. 1963–1968.

[17] K. Kant and S. W. Zucker, *Toward efficient trajectory planning: The path-velocity decomposition*, The international journal of robotics research, 5 (1986), pp. 72–89.

[18] N. R. Kapania, J. Subosits, and J. Christian Gerdes, *A sequential two-step algorithm for fast generation of vehicle racing trajectories*, Journal of Dynamic Systems, Measurement, and Control, 138 (2016), p. 091005.

[19] Y. Kim, J. Park, W. Son, and T. Yoon, *Modified turn algorithm for motion planning based on clothoid curve*, Electronics Letters, 53 (2017), pp. 1574–1576.

[20] S. Koenig and M. Likhachev, *D* lite*, in Eighteenth national conference on Artificial intelligence, 2002, pp. 476–483.

[21] J. Z. Kolter and A. Y. Ng, *Task-space trajectories via cubic spline optimization*, in 2009 IEEE international conference on robotics and automation, IEEE, 2009, pp. 1675–1682.

[22] E. Lambert, R. Romano, and D. Watling, *Optimal path planning with clothoid curves for passenger comfort*, in Proceedings of the 5th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2019), vol. 1, SciTePress, 2019, pp. 609–615.

[23] B. Lindqvist, S. S. Mansouri, A.-a. Agha-mohammadi, and G. Nikolakopoulos, *Nonlinear mpc for collision avoidance and control of uavs with dynamic obstacles*, IEEE robotics and automation letters, 5 (2020), pp. 6001–6008.

[24] C. S. Ma and R. H. Miller, *Milp optimal path planning for real-time applications*, in 2006 American Control Conference, IEEE, 2006, pp. 6–pp.

[25] D. G. Macharet, A. A. Neto, V. F. da Camara Neto, and M. F. Campos, *Nonholonomic path planning optimization for dubins' vehicles*, in 2011 IEEE International Conference on Robotics and Automation, IEEE, 2011, pp. 4208–4213.

[26] T. Maekawa, T. Noda, S. Tamura, T. Ozaki, and K.-i. Machida, *Curvature continuous path generation for autonomous vehicle using b-spline curves*, Computer-Aided Design, 42 (2010), pp. 350–359.

[27] M. Neunert, C. De Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli, *Fast nonlinear model predictive control for unified trajectory optimization and tracking*, in 2016 IEEE international conference on robotics and automation (ICRA), IEEE, 2016, pp. 1398–1404.

[28] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, *A survey of motion planning*

and control techniques for self-driving urban vehicles, IEEE Transactions on intelligent vehicles, 1 (2016), pp. 33–55.

[29] G. Parlangeli, L. Ostuni, L. Mancarella, and G. Indiveri, *A motion planning algorithm for smooth paths of bounded curvature and curvature derivative*, in 2009 17th Mediterranean Conference on Control and Automation, 2009, pp. 73–78.

[30] A. Ravankar, A. A. Ravankar, Y. Kobayashi, Y. Hoshino, and C.-C. Peng, *Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges*, Sensors, 18 (2018), p. 3170.

[31] C. Richter, A. Bry, and N. Roy, *Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments*, in Robotics Research: The 16th International Symposium ISRR, Springer, 2016, pp. 649–666.

[32] P. Scheffe, T. M. Henneken, M. Kloock, and B. Alrifaee, *Sequential convex programming methods for real-time optimal trajectory planning in autonomous vehicle racing*, IEEE Transactions on Intelligent Vehicles, 8 (2022), pp. 661–672.

[33] J. A. Silva and V. Grassi, *Clothoid-based global path planning for autonomous vehicles in urban scenarios*, in 2018 IEEE international conference on robotics and automation (ICRA), IEEE, 2018, pp. 4312–4318.

[34] K. R. Simba, N. Uchiyama, and S. Sano, *Real-time smooth trajectory generation for nonholonomic mobile robots using bézier curves*, Robotics and Computer-Integrated Manufacturing, 41 (2016), pp. 31–42.

[35] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, *OSQP: an operator splitting solver for quadratic programs*, Mathematical Programming Computation, 12 (2020), pp. 637–672, https://doi.org/10.1007/s12532-020-00179-2, https://doi.org/10.1007/s12532-020-00179-2.

[36] F. Stoican, A. Postolache, and I. Prodan, *Nurbs-based trajectory design for motion planning in a multi-obstacle environment*, in 2021 European Control Conference (ECC), IEEE, 2021, pp. 2014–2019.

[37] Z. Tahir, A. H. Qureshi, Y. Ayaz, and R. Nawaz, *Potentially guided bidirectionalized rrt* for fast optimal path planning in cluttered environments*, Robotics and Autonomous Systems, 108 (2018), pp. 13–27.

[38] G. Tang, C. Tang, C. Claramunt, X. Hu, and P. Zhou, *Geometric A-Star Algorithm: An Improved A-Star Algorithm for AGV Path Planning in a Port Environment*, IEEE Access, 9 (2021), pp. 59196–59210, https://doi.org/10.1109/ACCESS.2021.3070054.

[39] L. Xu, M. Cao, and B. Song, *A new approach to smooth path planning of mobile robot based on quartic bezier transition curve and improved pso algorithm*, Neurocomputing, 473 (2022), pp. 98–106.

[40] H. Xue, T. Yue, and J. M. Dolan, *Spline-based minimum-curvature trajectory optimization for autonomous racing*, arXiv preprint arXiv:2309.09186, (2023).

[41] X. Zhang, J. Ma, Z. Cheng, S. Huang, S. S. Ge, and T. H. Lee, *Trajectory generation by chance-constrained nonlinear mpc with probabilistic prediction*, IEEE Transactions on Cybernetics, 51 (2020), pp. 3616–3629.