IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 25, 2022, accepted February 13, 2022, date of publication February 16, 2022, date of current version February 25, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3152187

A Hierarchical Motion Planning Framework for Autonomous Driving in Structured Highway Environments

DONGCHAN KIM, GIHOON KIM, HAYOUNG KIM, AND KUNSOO HUH, (Member, IEEE)
Department of Automotive Engineering, Hanyang University, Seoul 04763, South Korea

Corresponding author: Kunsoo Huh (khuh2@hanyang.ac.kr)

This work was supported by the Ministry of Trade, Industry, and Energy (MOTIE), South Korea, through the Technology Innovation Program (Industrial Strategic Technology Development Program, Development of Test Procedure Standards for V2I Connected Automated Driving Systems) under Grant 20014460.

ABSTRACT This paper presents an efficient hierarchical motion planning framework with a long planning horizon for autonomous driving in structured environments. A 3D motion planning with time information is a non-convex problem because there exists more than one local minimum point and various constraints such as roads and obstacles. Accordingly, to deal with high computational complexity and the problem of falling into a local minimum in an enormous solution space, a decoupled method is utilized, that consists of two steps: *Long-term* planning and *short-term* planning. First, the *long-term* planner provides reasonable far-sighted behavior through two processes. In the first process, a rough path that includes a driving strategy is generated in the 2D spatial space. Then, the jump point search algorithm is utilized with time information on the path to reduce the computational burden of A*, giving an acceptable quality of solution at the same time. In this step, a safe, comfortable, and dynamically feasible trajectory is generated. Next, the *short-term* planner optimizes a short-sighted trajectory using particle swarm optimization. In this method, a steering angle set is encoded as a particle, resulting in a safe, comfortable, and kinodynamically feasible trajectory. The proposed algorithm is implemented and evaluated in a range of vehicle-in-the-loop simulation scenarios, which include various virtual static and dynamic obstacles generated by Intelligent Driver Model. In the evaluation results, the proposed method reduced the computation time by up to 0.696 s with increasing the step cost by up to about 3%. The proposed algorithm is executed every 100 ms for a planning horizon of 10 seconds, and the average computation time is 31 ms with the worst-case computation time of 94 ms.

INDEX TERMS Motion planning, Dijkstra's algorithm, jump point search algorithm, particle swarm optimization.

I. INTRODUCTION

During the last few decades, research on autonomous driving has been actively studied worldwide, both in academia and the industry [1], [2] to aid drivers and, reduce tedious driving-related tasks. It also aims to prevent accidents caused by carelessness of drivers, which accounts for the majority of casualties [3], [4]. As a core module and being the decision stage in autonomous driving, the motion-planning module generates a legal, safe, comfortable, and kinodynamically feasible trajectory considering the surrounding environments and vehicle states. To date, many motion planning algorithms have been developed.

The associate editor coordinating the review of this manuscript and approving it for publication was Zheng Chen.

A. RELATED WORK

The motion planning of an autonomous vehicle can be divided into two approaches [5]: Direct planning methods and decoupled planning methods based on whether the state space of the configuration space is decoupled.

In direct methods, the optimal solution is selected instantaneously through optimal planning using searching or optimization approaches in a spatiotemporal state space, which includes time information along with longitudinal and lateral motions. Ziegler *et al.* [6] used a method that transforms

a nonconvex problem into a convex problem considering the comfort, safety, and smoothness of the trajectory. Then, using a sequential quadratic programming (SQP) algorithm, a nonlinear optimization problem is solved numerically. It is well-known that the SQP algorithm should have a good initial value for an acceptable computation time [7]. Chen *et al.* proposed the constrained iterative linear-quadratic regulator (CILQR) to efficiently solve the optimal control problem with nonlinear system dynamics and a general form of constraints [8]. The computational efficiency of CILQR was shown to be much higher than that of the SQP solver. However, without a good initial value, it can also influence the convergence speed of the algorithm.

For other approaches of direct methods, spatiotemporal state lattice-based trajectory planning methods were introduced to generate a feasible trajectory in a dynamic scenario [9], [10]. The choice of resolution of the search space is a trade-off between computation time and solution quality. If the resolution increases, the computation time increases. If the resolution decreases, the optimality of the solution space may be degraded, and a feasible solution cannot be found. Dorff *et al.* proposed a trajectory planning and control method using nonlinear model predictive controller (NMPC) [11]. It is validated in partially occluded parking environments with safe planning horizon of 1 s to 1.5 s. In addition, in [12] and [13], MPC-based trajectory planning methods were used. MPC solves a sequence of finite-time optimization problems in a recursive manner and generates consecutive control actions regarding the vehicle's motion. A comfortable trajectory is generated while guaranteeing the safety of an autonomous vehicle. However, this method shows poor performance for non-convex and high-complexity problems.

Conversely, decoupled methods attempt to generate an optimal solution through multiple steps by reducing the dimensions of the state space. Werling *et al.* [14] generated longitudinal and lateral trajectories separately considering dynamic obstacles in the Frenet coordinate and then selected one optimal trajectory. Among the generated trajectories using quartic and quintic polynomials versus time, the jerk-minimizing trajectory is chosen. This method has the drawback of frequent swerving motion due to shortsighted planning. In [15] and [16], Li *et al.* performed a path-speed decomposition method to obtain an optimal trajectory step by step. First, a spatial path was generated using a curvature polynomial, and then a trapezoidal velocity profile was smoothed using a polynomial spline to obtain a trajectory. This method has the drawback of a short planning horizon and incompleteness of the solution owing to the finite set of motion primitives. In [17], a combination of dynamic programming and quadratic programming was proposed to generate path and speed profiles, respectively. A 3D station-lateral-speed problem is transformed into two 2D station-lateral and station-speed problems to reduce the complexity of the problem and computation time. Unfortunately, a limitation is that the generated trajectory is not guaranteed to be kinematically feasible. Xu *et al.* [18] used a method that consisted of two parts: Trajectory planning and trajectory optimization. First, a rough path and speed profile are generated, and then, they are iteratively optimized. However, non-holonomic constraints are not guaranteed to be satisfied. Zhang *et al.* proposed a method which adopts several steps [19]. In the first step, in a Frenet frame, a smooth driving guide line is obtained, then, optimization is performed for path generation. The second strategy is the proposition of piecewise-jerk path formulation. Finally, the optimization is performed to search a safe and kinematically feasible solution considering obstacles.

In addition, Li *et al.* proposed a semantic-level maneuver decision-making and trajectory planning method [20]. After the upper-level maneuver is decided, the lower-level trajectory planning is decoupled into longitudinal and lateral directions. Then, the selected trajectory is optimized by the operator splitting quadratic program (OSQP). This method is validated using Prescan simulator under two scenarios. Zhang *et al.* [21] introduced a hybrid trajectory planning method that generates a smooth, safe, and computationally efficient solution. However, only static obstacles were considered in their work. Lim *et al.* [22] used a hierarchical strategy in which a sampling-based approach was used for behavioral planning, and optimization was conducted for trajectory planning. However, the planner lacks the capability to generate trajectories with long horizons. Jin *et al.* [23] proposed a method to adaptively change the longitudinal horizons considering obstacles for better performance of on-road autonomous driving with avoidance of both static and moving obstacles. In decoupled methods, static obstacle is typically considered in the path planning stage and dynamic obstacle is considered in the speed planning stage. Thus, it is not guaranteed that the decoupled method is optimal. In addition, the number of tuning parameters is usually much more than the direct methods and their adjustment becomes difficult.

B. CONTRIBUTION

Nonetheless, a decoupled method was used for motion planning in our work to reduce the problem complexity and computational burden. Motion planning for self-driving can be divided into two parts [24], behavioral planning and trajectory planning. Behavioral planning can be categorized as high-level decision-making and is responsible for *long-term* planning ($t_{horizon} > 5$s) in complex situations [25]. Many studies have been conducted on state machines when performing *long-term* planning [26]–[30]. Various behaviors are represented as predefined states, and then an appropriate behavior is selected at every step. However, when the situation becomes complicated and the maneuvers to model are increased, it becomes computationally intractable owing to the exponentially growing number of state transition rules. Therefore, to address this problem, Hubmann *et al.* [25] proposed a generic *long-term* planning method using the A* graph search algorithm. However, only longitudinal planning was regarded in their work.

In this paper, a hierarchical framework for motion planning in structured environments is proposed that provides a *long-term* solution that considers both the longitudinal and lateral directions. The algorithm structure is divided into two steps, *long-term* planning and *short-term* planning. *Long-term* planning is performed first and consists of two processes. In the first process, an optimal spatial path is obtained by considering the road geometry and obstacles. This part determines which space to drive is desirable in a complex situation. In the next process, a spatiotemporal trajectory is generated by placing the position and velocity profiles versus time on the previously obtained *long-term* path, which has a reduced searching space. This process provides a legal, safe, comfortable, and dynamically feasible *long-term* maneuver. In the second step, *short-term* planning is performed using the front part of the *long-term* solution. This step provides a safe, comfortable, and kinodynamically feasible trajectory in a combined resampling and optimization manner. Finally, the performance of the proposed algorithm is verified through a vehicle-in-the-loop simulation (VILS) with a real car under various scenarios. The contributions of this study can be summarized as follows:

- The proposed hierarchical scheme can generate safe, comfortable, and kinodynamically feasible trajectories that can deal with static and dynamic obstacles.
- Jump point search (JPS) algorithm which has been used only for the problems with a uniform step cost, is utilized in our problem which has a non-uniform step cost with the help of a carefully designed assumption. Leveraging JPS algorithm reduces the computation time significantly and allows real-time long-term planning with an acceptable solution quality.
- The short-term trajectory planner optimizes the initial trajectory from the long-term planner. Furthermore, steering angle sets are encoded as particles in particle swarm optimization (PSO) through a kinematic vehicle model that satisfies the non-holonomic constraint. Additionally, in PSO, the driving comfort, safety, and dynamic constraints are considered as well.
- The feasibility of the proposed method was demonstrated in various automated driving scenarios using VILS.

C. PAPER ORGANIZATION

The remainder of this paper is organized as follows: The overall algorithm framework is introduced in Section II. The optimal *long-term* path planning using the shortest-path algorithm and *long-term* trajectory planning using a JPS algorithm are described in Section III. In Section IV, *short-term* trajectory optimization using PSO algorithm is explained. Section V explains the experimental setup and evaluation of the proposed algorithm in a vehicle-in-the-loop environment. Section VI summarizes the proposed algorithm and discusses the future work.

II. ALGORITHM FRAMEWORK

To generate a trajectory in an autonomous driving situation that is non-convex [31], the autonomous vehicle must consider complex driving environments which comprise static and dynamic surrounding vehicles, roads, traffic rules, etc. Based on the scene information, an autonomous vehicle must run within the normal speed range, change lanes, and drive safely without colliding with surrounding vehicles. Therefore, when a trajectory is generated, safety, comfort, and limits of the vehicle kinematics and dynamics should be considered. The overall algorithm framework for optimal motion planning is shown in Fig. 1. This study focuses on a hierarchical motion planning framework for autonomous driving on unidirectional roads. The information of the surrounding vehicle states from the perception module is assumed to be available to the planning module.

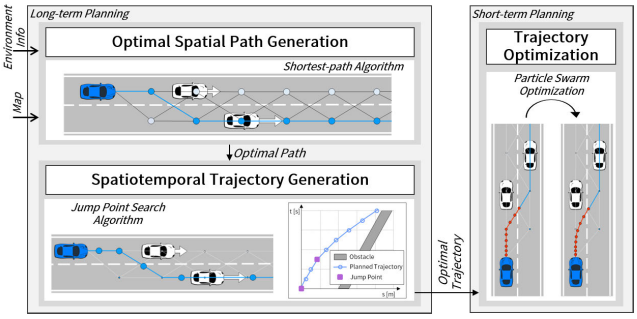

FIGURE 1. Overall hierarchical algorithm framework for optimal motion planning.

In the first step, the *long-term* ($\sim 10s$) spatial path including the longitudinal and lateral motions, is obtained by considering the surrounding vehicles. A maneuver with a lane-level lateral motion target is generated by considering the current state and prediction during a specific time horizon of the surrounding environments.

In the second step, the position and velocity profiles are applied to the optimal spatial path from the first step to obtain a *long-term* spatiotemporal trajectory. Safe motion planning is carried out in which a collision-free trajectory is generated considering static and dynamic obstacles. The improved A* algorithm, JPS, was utilized as the base algorithm to reduce the computational effort [32]. The original JPS is optimized for solving the grid-based problem, which has a uniform cost for each action. In our problem, which is without a uniform cost for each action, a sub-optimality condition is defined and the JPS is applied.

Finally, *short-term* trajectory optimization was performed using the front-part of the optimal *long-term* trajectory. It is of practical importance to follow the adjacent *short-term* ($\sim 3s$) trajectory when tracking the control of the generated trajectory. In our work, the PSO algorithm was utilized to make the *short-term* trajectory kinodynamically feasible, smooth, and safe.

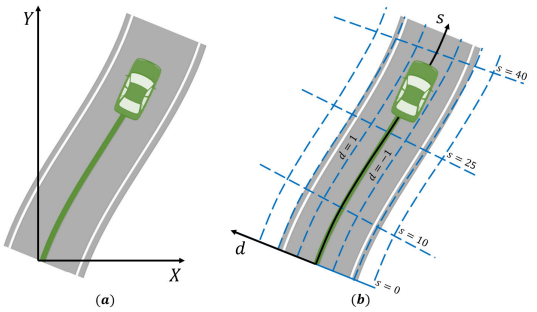

FIGURE 2. Frenet coordinates of the road geometry. (a) Cartesian coordinates (b) Frenet coordinates.

III. LONG-TERM PLANNING

In this section, the concept of *long-term* planning of the proposed hierarchical framework is explained in detail. First, the optimal spatial path generation is described, then the spatiotemporal trajectory using the optimal spatial path is described.

A. OPTIMAL SPATIAL PATH GENERATION

To create an optimal spatial path, a search space is created that comprise lane-level nodes. An optimal path selection process was performed on the search space. Each node is represented by the following 3D space as state n.

$$\mathrm{N} = \left\{ n \equiv [s, d, l]^T | s \in \mathbb{R}, d \in \mathbb{R}, l \in [1, N_l] \right\} \tag{1}$$

where s and d are defined as the longitudinal and lateral positions along the road, respectively, l is defined as the rightmost lane having a value of 1 and the leftmost lane having a value of N_l. The road geometry was defined according to the Frenet coordinates, as described in Fig. 2. The search space consists of a total of $N_p \times N_l$ nodes in the Frenet coordinates. In addition, except for the initial node n_0, the k^{th} step longitudinal position s_k of node n_k is expressed as follows:

$$\Delta s = v_0 \Delta t_{lt} \tag{2}$$

$$s_k = s_0 + k\Delta s \tag{3}$$

where v_o and s_0 are the initial velocity and longitudinal position of the ego vehicle, respectively, and Δt_{lt} is the time interval. Δs is determined by the current speed of the ego vehicle. However, to prevent the search space from disappearing under low speeds, Δs is set to be greater than Δs_{min}, which is set to 10 m in our work.

Fig. 3 shows an example of the optimal spatial path generation. The blue vehicle is the ego autonomous vehicle and the white vehicle represents the obstacle vehicle during the prediction horizon. The nodes were placed in the direction of the road with an interval of Δs. Each node is generated at every layer, and the result is defined as a directed, acyclic graph. The edge is a linear path from a node in one layer to a node in the next layer. The cost of edge $e_{n_k \to n_{k+1}}$ is shown

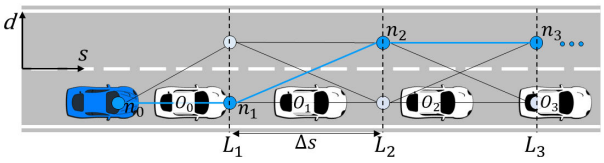

FIGURE 3. Example of the optimal spatial path generation.

in the following equation, where n_k is a node in layer L_k and n_{k+1} is a node in layer L_{k+1}.

$$J(e_{n_k \to n_{k+1}}) = w_{dist} dist(e_{n_k \to n_{k+1}}) + w_c c(e_{n_k \to n_{k+1}}) + w_{col} col(e_{n_k \to n_{k+1}}) \tag{4}$$

where w_{dist}, w_c, w_{col} are weights for each index.

The first cost term represents the length of each edge using the Euclidean distance between two nodes, and it prevents the ego vehicle from changing lanes frequently by assigning a penalty to a long edge, as shown in (5).

$$dist\left(e_{n_k \to n_{k+1}}\right) = \left| \overrightarrow{n_k(1,2) n_{k+1}(1,2)} \right| \tag{5}$$

The second cost term is for the consistency of consecutive plans. This term reduces the difference in the lateral offset between the current path candidate and the previous optimal spatial path, as described in (6):

$$c\left(e_{n_k \to n_{k+1}}\right) = \left| d_{k+1}^{prev} - d_{k+1} \right| \tag{6}$$

where d_{k+1}^{prev} and d_{k+1} are the lateral positions of the previously selected optimal path and a new path candidate, respectively.

The velocity profile of the previous planning cycle was used to calculate the predicted position of the ego vehicle. It is assumed that using the velocity profile of the previous planning cycle is reasonable because the planning frequency is sufficiently high. The last collision cost term suppresses the collision between the predicted ego vehicle and the surrounding vehicles predicted with the constant velocity (CV) model and is defined as

$$col(e_{n_k \to n_{k+1}}) = \begin{cases} c_{col}, & \text{if } \left| s_{e,i} - s^{O_i^j} \right| < s_{sf} \text{ and } l_i = l^{O_i^j} \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

$$\text{with} \quad s_{sf} = \frac{L_O}{2} + \frac{L_{ego}}{2} + s_{margin}. \tag{8}$$

Here, subscript j denotes a number from 1 to N_o where N_o is the number of obstacles in the region of interest. For each j from $i = k_s$ to $i = k_c$, it is checked if the condition in (7) is satisfied. If the condition is satisfied even once, cost c_{col} is imposed. Furthermore, k_s and k_e represent the start and end indices of the s-axis position, where the predicted ego vehicle is within the edge $e_{n_k \to n_{k+1}}$, respectively. Additionally, $s_{e,i}$ and $s^{O_i^j}$ are the s-axis positions of the predicted ego vehicle and j^{th} obstacle O_i^j at the i^{th} step, respectively.

Further, l_i and $l^{O_i^j}$ are the lane values of nodes n_i and j^{th} obstacle, O_i^j, at the i^{th} step, respectively. A constant value c_{col} is set sufficiently large so as to make it dominate the overall cost function for penalizing collisions. The lengths of the obstacle and ego vehicle are L_O and L_{ego}, respectively. The margin distance value is s_{margin}, and s_{sf} is the safety distance to restrict collisions between vehicles.

This becomes a shortest-path problem, in which dynamic programming is utilized to solve it quickly. The shortest-path problem is an algorithm for finding the shortest path between nodes in a graph. In our problem, one node per layer is selected to construct the path using the cost function in (4). The optimal solution with minimum total cost is given by a sequence of nodes on the N_p layers covering the space that satisfies:

$$\underset{\{e_{n_k \to n_{k+1}}\}}{argmin} \sum_{k=0}^{N_p-1} J(e_{n_k \to n_{k+1}}). \tag{9}$$

B. SPATIOTEMPORAL TRAJECTORY GENERATION PROBLEM FORMULATION

A spatial path planner provides the desired maneuver to a spatiotemporal trajectory planner considering the *long-term* objective of autonomous driving in the spatial search space. An optimization problem is formulated to generate a trajectory based on the optimal spatial path given, and is represented as:

$$\begin{aligned}
&min\ J = c_{tot} = \sum_{k=0}^{N_p-1} c\,(x_k, a_k, x_{k+1}, O_{ol}) \\
&s.t.\ x_{k+1} = A \begin{bmatrix} x_k \\ 1 \end{bmatrix} + Ba_k \\
&\quad c(\cdot) = w_v c_v(x_{k+1}) + w_A c_A(a_k) + c_O(x_k, x_{k+1}, O_{ol}) \\
&\quad O_{ol} = \left\{ \left[l^O_{len}, s^O_{ol,s,i}, s^O_{ol,e,i}, t^O_{ol,s,i}, t^O_{ol,e,i} \right]^T | i = 1 \cdots m \right\} \\
&\quad O = \{ O_k | k = 0 \cdots N_p \} \\
&\quad a_k \in \mathcal{A},
\end{aligned} \tag{10}$$

The detailed description is contained in Sections III-B1 to III-B3. State x in the state space $\mathcal{X} \subseteq \mathbb{R}^3$ is represented as follows:

$$\mathcal{X} = \left\{ x \equiv [s, v, t]^T | s \in \mathbb{R}^+, v \in \mathbb{R}^+, t \in \mathbb{R}^+ \right\} \tag{11}$$

where the state x includes the longitudinal position s, velocity v and time t.

The problem is formulated as a discrete planning problem [33] with a similar approach as in [25] and is solved using the JPS algorithm, which will be explained in detail in Section III-C.

1) TRANSITION MODEL

The state x_k represents a state in the planning step k, and the state transition model in discretized form is expressed as follows:

$$\begin{aligned}
x_{k+1} = \begin{bmatrix} s_{k+1} \\ v_{k+1} \\ t_{k+1} \end{bmatrix} &= \begin{bmatrix} 1 & \Delta t_{lt} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t_{lt} \end{bmatrix} \begin{bmatrix} s_k \\ v_k \\ t_k \end{bmatrix} \\
&+ \begin{bmatrix} 0.5(\Delta t_{lt})^2 \\ \Delta t_{lt} \\ 0 \end{bmatrix} a_k
\end{aligned} \tag{12}$$

where a_k is an acceleration candidate belonging to a discretized acceleration set $\mathcal{A}$. The state is expanded using the state transition model for a limited time T_{lt} with an interval of Δt_{lt}.

2) REPRESENTATION OF THE ENVIRONMENT

The autonomous vehicle encounters a static and dynamic obstacle O on the road, the information of a specific obstacle at the k^{th} step is parameterized as a vector, $\left[l^{O_k}_{len}, s^{O_k}, d^{O_k} \right]^T$ where $l^{O_k}_{len}$ is the length of the obstacle, s^{O_k} is the longitudinal position, and d^{O_k} is the lateral position.

Next, a procedure is performed to extract information about the obstacles that affect the ego vehicle. The *long-term* optimal spatial path obtained is checked to determine the extent of overlap with the prediction of surrounding vehicles. The overlapped obstacle O_{ol} is represented as follows using O which includes the accumulated information of O_k during the prediction horizon:

$$O_{ol} = \left\{ \left[l^O_{len}, s^O_{ol,s,i}, s^O_{ol,e,i}, t^O_{ol,s,i}, t^O_{ol,e,i} \right]^T | i = 1 \cdots m \right\} \tag{13}$$

$$O = \{ O_k | k = 0 \cdots N_p \} \tag{14}$$

where $s^O_{ol,s,i}$ and $s^O_{ol,e,i}$ represent the start and end positions in the longitudinal direction of the optimal spatial path at the i^{th} overlap with the specific obstacle, respectively. Furthermore, $t^O_{ol,s,i}$ and $t^O_{ol,e,i}$ represent the time information at the corresponding step, respectively.

Fig. 4 shows an example in which one obstacle is overlapped twice on the optimal spatial path. In Fig. 4(a), the predicted ego vehicle (light blue color) through the optimal spatial path (blue solid line) represents the position information for the N_p step calculated from the velocity profile of the previous planning cycle.

In Fig. 4(b), the overlapped information is expressed as $\left\{ \left[l^O_{len}, s^O_{ol,s,i}, s^O_{ol,e,i}, t^O_{ol,s,i}, t^O_{ol,e,i} \right]^T | i = 1, 2 \right\}$ using (13) where $m = 2$. It is considered overlapped if the prediction of the surrounding vehicle is in the same lane as the optimal spatial path, and the difference in the longitudinal position between the obstacle and the ego vehicle is within s_{sf}. Assuming highway driving, the variation in s_{sf} owing to the heading angle is ignored. The overlapped path is represented by the red line in Fig. 4(b).

At $k = 1$ and $k = 2$, the ego vehicle and obstacle overlap. Therefore, it is assumed that the position of s between the two points continues to overlap. Next, at $k = 3$, the optimal

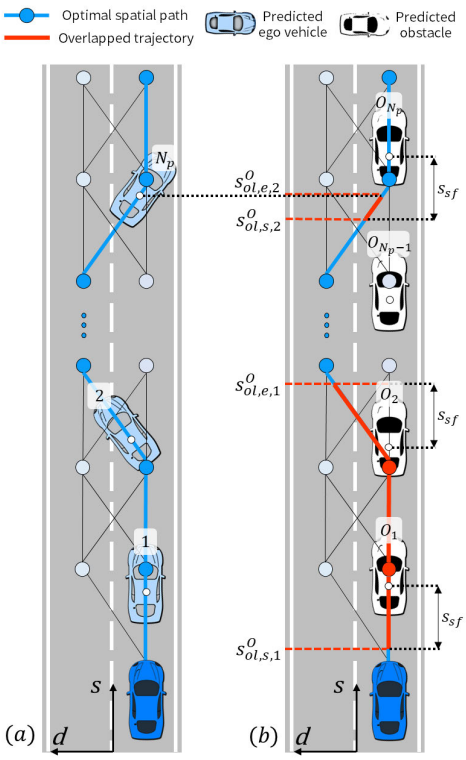

FIGURE 4. Example of overlapped information throughout the optimal spatial path.

spatial path moves to the left lane and does not overlap with the obstacle at the same time step. Therefore, the first overlap section is set as: $[s^O_{ol,s,1}, s^O_{ol,e,1}]$.

Next, at $k = N_p - 1$, the optimal spatial path and obstacle do not overlap, whereas at $k = N_p$, they overlap. In this case, utilizing the s positions of the ego vehicle, s position of the obstacle and s_{sf}, the second overlap section is set as $[s^O_{ol,s,2}, s^O_{ol,e,2}]$. Furthermore, $t^O_{ol,s,1}, t^O_{ol,e,1}, t^O_{ol,s,2}, t^O_{ol,e,2}$ represent time information corresponding to each s overlap value.

3) COST FUNCTION

If an action a_k is taken to state x_k, it moves to the next state x_{k+1} through the transition model. The step cost $c(x_k, a_k, x_{k+1}, O_{ol})$ is defined for each path, and the overall cost c_{tot} which is summed along the path is represented as follows [33]:

$$c_{tot} = \sum_{k=0}^{N_p-1} c\,(x_k, a_k, x_{k+1}, O_{ol}) \tag{15}$$

The path with a minimal cost from the initial state to the goal state is obtained using the graph search algorithm. The step cost is defined as

$$\begin{aligned} &c\,(x_k, a_k, x_{k+1}, O_{ol}) \\ &\quad = w_v c_v(x_{k+1}) \\ &\qquad + w_A c_A(a_k) + c_O(x_k, x_{k+1}, O_{ol}) \end{aligned} \tag{16}$$

where w_v and w_A are weights for each index.

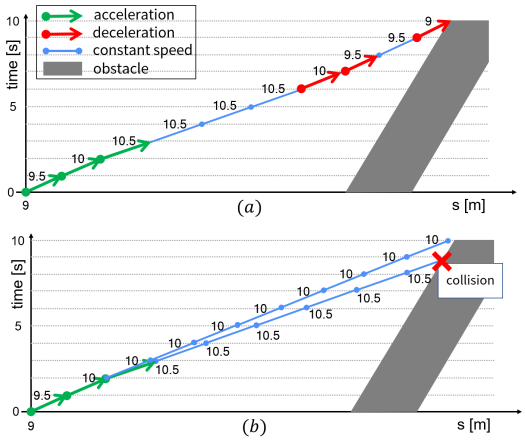

FIGURE 5. Example of solutions compared for a target velocity of 20 m/s using (a) an optimal solution with A* (b) a suboptimal solution with JPS.

The first cost term represents the deviation from the reference velocity v_{ref} and is defined as follows:

$$c_v(x_{k+1}) = \left(\frac{v_{k+1} - v_{ref}(s_{k+1})}{v_{ref}(s_{k+1})}\right)^2. \tag{17}$$

The limit velocity $v_{limit} = \sqrt{\frac{\ddot{d}_{max}}{\kappa}}$ is determined by the maximum lateral acceleration $\ddot{d}_{max}$ and road curvature κ. Furthermore, there exists a road velocity limit v_{road}. v_{ref} is decided to be the minimum value of these two velocities:

$$v_{ref} = min(v_{limit}, v_{road}). \tag{18}$$

The second cost term penalizes the size of the action that belongs to acceleration set $\mathcal{A}$. It increases the driver's comfort by providing cost in the form of squared acceleration.

The last cost term represents the cost of obstacles. An obstacle has an area on the distance-time plane in state space $\mathcal{X}$. The cost for the static and dynamic obstacles is expressed as follows using the overlapped obstacle O_{ol}:

$$c_O(x_k, x_{k+1}, O_{ol}) = c_{O_{ol}}(x_k, x_{k+1}), \tag{19}$$

$$\text{with} \quad c_{O_{ol}}(x_k, x_{k+1}) = \begin{cases} inf, & \text{if } \{\overrightarrow{x_k x_{k+1}}\} \cap O_{ol} \neq \emptyset \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

where $\overrightarrow{x_k x_{k+1}}$ is defined as the set of points on the vector between x_k and x_{k+1}.

C. SPATIOTEMPORAL TRAJECTORY GENERATION PROBLEM SOLVING

Many search algorithms exist to solve planning problems. Dijkstra, A*, and JPS are widely used in discrete planning problems. The JPS algorithm [32] reduces the computation time of A* by adding only the jump points to the *open list*. The optimality of JPS is guaranteed only when the step cost is uniform, which is not the case in our study. Nonetheless, to take advantage of JPS in terms of computational efficiency

Algorithm 1 Function JPS

Require: x_s : start state, g : goal state, $h(x)$: heuristic, O : obstacle

$\mathbb{C} \leftarrow \emptyset$
$N = \{x_s, 0, 0, 0, \emptyset\}$
$\mathbb{O} \leftarrow N$
KeepSearching $\leftarrow 1$
while *KeepSearching* is 1 **do**
  if $\mathbb{O}$ is empty **then**
    $x_{new} = transition(N.x, -a_{max})$
    $N_{new} = \{x_{new}, c_{new}, h(x_{new}), -a_{max}, N\}$
    Add N_{new} to $\mathbb{O}$
  else
    N = *Extract* node with minimum $c + h$ from $\mathbb{O}$
    Add N to $\mathbb{C}$ and *Delete* from $\mathbb{O}$
    if N is g **then**
      trajectory = *Backtracking(N)*
      KeepSearching $\leftarrow 0$
    else
      foreach $a \in$Available action(N) **do**
        $\{x_{new}, c_{new}\} \leftarrow jump(N.x, a, g, O, N.c)$
        $N_{new} = \{x_{new}, c_{new}, h(x_{new}), a, N\}$
        if $N_{new} \in \mathbb{C}$ **then**
          continue
        else if N_{new} has the same state with $n \in \mathbb{O}$ with smaller cost **then**
          Replace n with N_{new}
        else
          Add N_{new} to $\mathbb{O}$
        end if
    end if
  end if
end while
return *trajectory*

Algorithm 2 Function Jump

Require: x_k : initial state, a_k : action, g : goal state, O : Obstacle, C_{cum} : cumulative cost

$x_{k+1} \leftarrow transition\,(x_k, a_k)$
if x_{k+1} is inside an obstacle **then** ▷ collision
  return $x_{k+1}, c\,(x_k, a_k, x_{k+1}, O) + C_{cum}$
else if $x_{k+1} = g$ **then** ▷ goal state
  return $x_{k+1}, c\,(x_k, a_k, x_{k+1}, O) + C_{cum}$
else if t_k is near end of $t^O_{ol,e,i}$ for any $o \in O$ **then** ▷ forced neighbor
  return $x_{k+1}, c\,(x_k, a_k, x_{k+1}, O) + C_{cum}$
else if $a \neq 0$ **then**
  for $a = 0$ **do**
    if *jump* (x_{k+1}, a_k, g) has finite cost **then**
      return $x_{k+1}, c\,(x_k, a_k, x_{k+1}, O) + C_{cum}$
    end if
  end for
else
  return *jump* (x_{k+1}, a_k, g) , $c\,(x_k, a_k, x_{k+1}, O) + C_{cum}$
end if

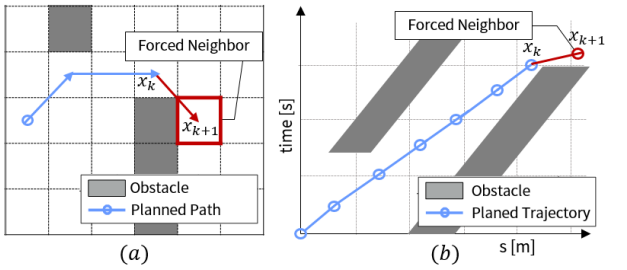

FIGURE 6. Forced neighbor. (a) Grid world (b) our problem.

and to carry out long-term planning in real time, the condition for the sub-optimality of the solution is defined as follows:

- In the time domain where obstacles overlap within the planning horizon of the ego vehicle, if $a = 0$ is taken once as the minimum action, only that action can be taken for the rest of the time.

A suboptimal solution is equivalent to the optimal solution if there are no other vehicles on the path, and it further, shows different movements if there are obstacles within the planning horizon of the ego vehicle. Let us assume that both the ego vehicle and the preceding vehicle on the path have a lower velocity than the reference velocity. In this case, as shown in Fig. 5(a), the optimal solution when w_A is low as in our case is as follows: first to accelerate to reach the target velocity of 20 m/s in the example, and then to decelerate near the obstacle. In contrast, as shown in Fig. 5(b), the suboptimal solution is to maintain $a = 0$ after acceleration. If it collides with the obstacle, as shown in Fig. 5(b), the node lastly accelerated is not added to the *open list*. Thereafter, the trajectory that maintains a constant speed is chosen as the suboptimal solution.

To an original optimal problem, JPS cannot be utilized because the minimum action that has a minimum step cost is not consistent and changes according to the state and action. However, to solve a suboptimal problem, as in our case, JPS can be utilized because maintaining a constant speed at last is a solution, and the constant speed ($a = 0$) can be interpreted as the minimum action.

Searching begins with the addition of the current state and zero cost to the *open list*. The state (x_k) with the lowest cost is excluded from the *open list*, and added to the *closed list*. If x_k is the goal state, then the search ends. If not, the jump point is searched within the action set which is modified to prevent the velocity of the next state from being lower than zero. If the searched state (x_{k+1}) is in the *open list*, but the sum of the step cost and heuristic is lower than that of the state in the list, the information of the corresponding state is updated. If the state (x_{k+1}) is not in the *open list*, it is added to the *open list*. This process is repeated for the state in the *open list*, and if the selected state is the goal, the

search is finished and the optimal profile is constructed by backtracking via the *closed list*. If the *open list* is empty as no state candidate exists, emergency braking with $a_{max} = 8$ is conducted for safety, as added in Lines 7 to 9. The pseudocode is described in Algorithm 1, where N represents a node consisting of $\{state, action, path\ cost, heuristic, parent\ node\}$. Furthermore, $\mathbb{O}$ and $\mathbb{C}$ indicate the *open list* and *closed list*, respectively.

In Line 18 of Algorithm 1, the JPS algorithm is performed as described in Algorithm 2, which represents a recursive function that makes a jump between the nodes. The jump step continues until the stop conditions are met [25], which consist of a collision, goal state, and forced neighbor as described in Lines 2 to 7 of the algorithm. In Line 1, the transition model described in (12) is utilized to obtain the next state, x_{k+1}, using the current state and action. Line 2 indicates the collision situation, Line 4 is when the goal state is reached and Line 6 is when the forced neighbor is encountered. The forced neighbor is represented as x_{k+1} in Fig. 6, which is the state blocked by obstacles and must be passed through x_k [32]. Lines 8 to 11 proceed when the first step's action is not the minimum action. In this case, the jump function recurses with the minimum action and checks if it collides with obstacles. If no collision occurs, the cost has a finite value, and the next state x_{k+1} is returned. Otherwise, the jump is performed again with the same step action, a_k. The search heuristic used in this study is represented as follows:

$$h = w_v \sum_{k=t}^{T_{lt}-1} \left(\frac{v_{k+1} - v_{ref}(s_{k+1})}{v_{ref}(s_{k+1})} \right)^2, \tag{21}$$

where

$$v_{k+1} = v_k + \Delta t_{lt} \underset{a \in A}{\operatorname{argmin}} (v_{ref}(s_{k+1}) - (v_k + a\Delta t_{lt})). \tag{22}$$

In (22), the action that minimizes the difference between the reference velocity and the state's velocity at $k+1$ is calculated. Then, the selected action is applied to obtain v_{k+1}, which is utilized in (21). This process is repeated from $k = t$ until $k = T_{lt} - 1$. The heuristic is admissible in (21) and has exactly the same form as the cost c_v in the step cost in (16), and it is guaranteed to have a lower value than the minimal cost from $k = t$ to $k = T_{lt} - 1$ which is expressed as $\sum_{k=t}^{T_{lt}-1} c^*(x_k, a_k, x_{k+1}, O_{ol})$. In addition, as long as the heuristic is the lower bound on the actual cost and is consistent, the A* algorithm is complete. Consistency is fulfilled if $h(x_k) \leq c(x_k, a_k, x_{k+1}, O_{ol}) + h(x_{k+1}) \forall x_k, x_{k+1}$. Thus, an admissible and consistent heuristic was used in this study.

Next, to verify the performance of the solution using the JPS, three metrics which comprise the number of nodes in the *open list*, computation time, and step cost are evaluated under the test scenarios generated in Section V. In Table 1, the metrics are compared with the JPS for the test scenarios. The average value was used to evaluate the first two metrics, and the ratio to the JPS was utilized for the last metric. For the number of nodes in the *open list*, it is significantly reduced for all scenarios. Accordingly, the computation time was also greatly reduced for each test scenario. Finally, in the worst-case scenario related to the step cost, the optimal solution from A* reaches 0.97 times that of the solution from the JPS, which is acceptable.

TABLE 1. Quantitative comparison of evaluation metrics.

		w/o h	A* w/ h	JPS
# of nodes in *open list* [-]	Scen. 1	695.0	665.0	81.6
	Scen. 2	1883.8	1282.2	529.4
	Scen. 3	2355.3	1823.8	477.24
	Scen. 4	415.6	358.2	89.4
	Scen. 5	217.8	185.1	64.5
comp. time [s]	Scen. 1	0.033	0.032	0.003
	Scen. 2	0.358	0.157	0.036
	Scen. 3	0.727	0.346	0.031
	Scen. 4	0.028	0.023	0.005
	Scen. 5	0.007	0.005	0.003
step cost [-] (ratio to JPS)	Scen. 1	0.97	0.97	1.00
	Scen. 2	0.98	0.98	1.00
	Scen. 3	0.99	0.99	1.00
	Scen. 4	0.99	0.99	1.00
	Scen. 5	0.99	0.99	1.00

IV. SHORT-TERM PLANNING

In this section, the *short-term* planning of the proposed hierarchical framework is explained. First, to increase the resolution, resampling of the spatiotemporal trajectory obtained from *long-term* planning is performed. Then, the part that optimizes the trajectory by encoding a steering angle set as a particle is explained using the resampled trajectory as the reference position value in the PSO.

A. RESAMPLING OVER TRAJECTORY

The spatiotemporal trajectory obtained from the *long-term* planning is a behavioral strategy with a long time horizon. However, tracking a *short-term* trajectory is important for the control module, therefore, the front part of the entire *long-term* trajectory was used for the resampling step. The resultant trajectory has sufficient resolution for considering the continuous changes of the vehicle state. Linear interpolation between discrete nodes is used for the resampling step. The planning horizon of T_{st} is discretized by Δt_{st} resulting in the N_{pso} number of sample nodes.

B. TRAJECTORY OPTIMIZATION

PSO is an algorithm that was inspired by the flocking behavior of swarms [34]. It is well-known that PSO performs well when solving nonlinear optimization problems and has the advantages of a fast convergence speed with only a small number of parameters that need to be adjusted [35].

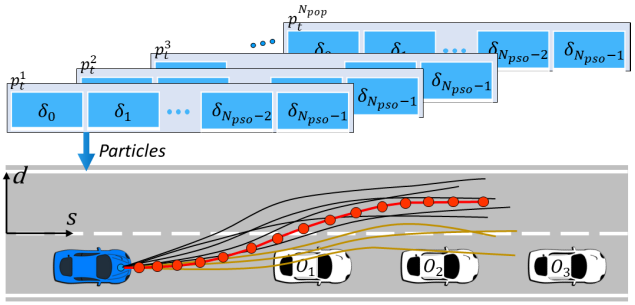

FIGURE 7. Example for a particle swarm encoding with kinodynamic constraints. Among the valid trajectories (black) except for invalid (yellow) trajectories, the optimal trajectory (red) is selected.

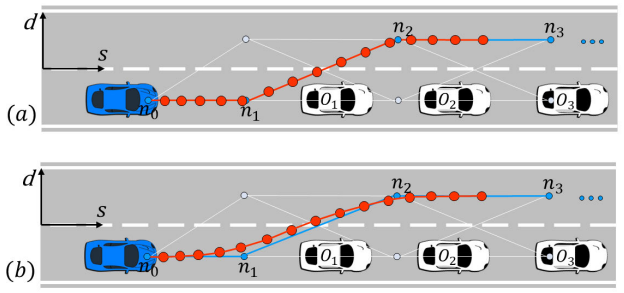

FIGURE 8. (a) Resampled behavioral trajectory. (b) Optimized *short-term* trajectory using the PSO algorithm.

In addition, it is a very efficient global search algorithm and derivative free. Furthermore, although there is no guarantee that PSO provides the global optimal solution, it is less likely to converge to a local minimum and finds good quality of solution with reasonable computational effort. For this reason, many studies have used the PSO for path planning [35]–[38].

Each particle p_t^i at time t in the swarm indicates a possible solution in the multidimensional search space. A particle consists of a total of N_{pso} steering angle values and becomes an N_{pso}-dimensional vector. The fitness of the particle is calculated through a designed fitness function, which is described in detail in Section IV-C. Each particle moves with a speed vector v_t^i, and the trajectory is adjusted accordingly. In addition, the personal best solution p_{pbest}^i and swarm's best solution p_{gbest} are updated respectively. Each particle is updated as follows:

$$p_{t+1} = v_{t+1} + p_t \tag{23}$$

Then, the velocity of each particle is updated considering the current velocity, the personal best solution x_{pbest}, and the global (the swarm's) best solution x_{gbest}, as follows:

$$u_{t+1} = w_i u_t + w_p r_1 (p_{pbest} - p_t) + w_g r_2 (p_{gbest} - p_t) \tag{24}$$

where w_i is the inertia weight damping ratio, w_p and w_g are the personal learning coefficient and global learning coefficient, respectively. Furthermore, r_1 and r_2 are uniformly distributed random numbers.

The optimization phase is terminated when the given optimization criteria are met, or if the maximum iteration limit is reached.

Fig. 7 shows an example of a steering angle set encoding as a particle of the PSO algorithm, and the i^{th} particle p_t^i, is expressed as shown in (25).

$$p_t^i = \left[\delta_0, \delta_1, \cdots, \delta_{N_{pso}-1}\right]^T \tag{25}$$

A particle consists of a steering angle set of the ego vehicle during horizon N_{pso}. A particle is passed through the kinematic [39], [40] that satisfies the non-holonomic constraint to obtain position information s_k, d_k, and heading angle θ_k as follows:

$$s_k = \Delta t_{st} \cdot v_{k-1} \cdot \cos(\theta_{k-1}) + s_{k-1} \tag{26}$$

$$d_k = \Delta t_{st} \cdot v_{k-1} \cdot \sin(\theta_{k-1}) + d_{k-1} \tag{27}$$

$$\theta_k = \Delta t_{st} \cdot \frac{v_{k-1}}{L} \cdot \tan(\delta_{k-1}) + \theta_{k-1} \tag{28}$$

where L is the length of the ego vehicle, and the subscript k denotes a number from 1 to N_{pso}. The obtained position set ensures that the curvature of the trajectory is within the kinematic constraints.

C. FITNESS FUNCTION

In the PSO, particles move in the direction of minimizing the fitness function to find the optimal value. In this study, the fitness function is composed of six terms and is expressed by the following equation.

$$f_{pso} = w_{ref} f_{ref} + w_{as} f_{as} + w_{ad} f_{ad} + w_{js} f_{js} + w_{jd} f_{jd} + f_{col} \tag{29}$$

where w_{ref} ,w_{as},w_{ad},w_{js} and w_{jd} are the weights for each index except for the last collision cost.

The first fitness term represents a reference cost and is defined as

$$f_{ref} = \sum_{i=1}^{N_{pso}} \left(s_i - s_{i,ref}\right)^2 + \left(d_i - d_{i,ref}\right)^2 \tag{30}$$

where $s_{i,ref}$ and $d_{i,ref}$ are the reference nodes for the longitudinal and lateral directions from the resampled behavioral trajectory. This fitness term makes a particle vector from the PSO that does not deviate significantly from the generated trajectory through the node and edge of the road geometry.

The next four fitness terms indicate the acceleration and jerk costs and are related to driving comfort, which are defined as follows:

$$f_{as} = \sum_{i=0}^{N_{pso}-2} a_{s,i}^2 = \sum_{i=0}^{N_{pso}-2} \left(\frac{s_{i+2} - 2s_{i+1} + s_i}{{\Delta t_{st}}^2}\right)^2 \tag{31}$$

$$f_{ad} = \sum_{i=0}^{N_{pso}-2} a_{d,i}^2 = \sum_{i=0}^{N_{pso}-2} \left(\frac{d_{i+2} - 2d_{i+1} + d_i}{{\Delta t_{st}}^2}\right)^2 \tag{32}$$

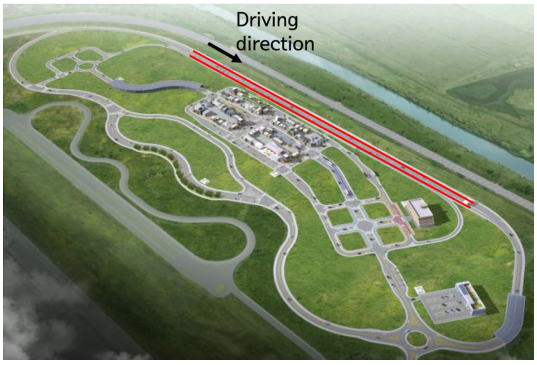

FIGURE 9. Test area: TS KATRI PG (Proving Ground).

$$f_{js} = \sum_{i=0}^{N_{pso}-3} j_{s,i}^2 = \sum_{i=0}^{N_{pso}-3} \left(\frac{-s_{i+3} + 3s_{i+2} - 3s_{i+1} + s_i}{{\Delta t_{st}}^3} \right)^2 \quad (33)$$

$$f_{jd} = \sum_{i=0}^{N_{pso}-3} j_{d,i}^2 = \sum_{i=0}^{N_{pso}-3} \left(\frac{-d_{i+3} + 3d_{i+2} - 3d_{i+1} + d_i}{{\Delta t_{st}}^3} \right)^2. \quad (34)$$

The acceleration and jerk values were calculated using forward finite differences. The physical limits of the vehicle dynamics for safe driving are considered using a circle of forces [41]. The acceleration values used when calculating the acceleration costs f_{as} and f_{ad} were constrained to be within the maximum acceleration:

$$a_{s,i}^2 + a_{d,i}^2 \leq a_{max}^2. \quad (35)$$

If the condition in (35) is violated, the i^{th} cost of f_a is set to be infinite to prevent the particle from being selected. The restriction check is not performed in Cartesian coordinates but rather in the Frenet space.

The last fitness term is related to collisions. The vehicle shape is represented using two circles for collision checks, similar as in [6],: wherein the distance between the centers of each circle of obstacles and the ego vehicle is required to be greater than the sum of each circle's radius. When the requirements are violated, f_{col} becomes infinite. In Fig. 8(a), the resampled trajectory is shown, and in Fig. 8(b), the initial trajectory is optimized using the PSO algorithm, which generates a kinodynamically feasible, smooth, and safe trajectory.

V. VEHICLE-IN-THE-LOOP SIMULATION

The VILS [42], [43] is implemented on the TS KATRI PG track, as shown in Fig. 9 with virtual scenarios and a test vehicle equipped with GPS, wherein it is assumed that information regarding virtual surrounding obstacles' positions and velocities is perfectly known. The proposed hierarchical motion planning framework is implemented in C++ language on a PC running Ubuntu 16.04 equipped with a quad-core Intel Core i7-6700K CPU. The execution period of the algorithm was 100ms. For validation, five highway scenarios in a unidirectional road were generated. Our first four scenarios included static and dynamic obstacles modelled by intelligent driver model (IDM). The last scenario included an obstacle performing a sudden cut-in maneuver. These scenarios were selected to show that our algorithm can perform many functionalities such as lane changing, car-following, obstacle avoidance, and stopping in various speed ranges. The motion planner parameters for implementation are listed in Table 2.

TABLE 2. Motion planner parameters.

Parameter	Value	Unit	Parameter	Value	Unit
N_l	2	-	Δt_{st}	0.1	s
T_{lt}	10	s	N_{pso}	30	-
Δt_{lt}	1	s	w_i	0.7	-
N_p	10	-	w_p	2	-
w_{dist}	100	-	w_g	2	-
w_c	10	-	w_{ref}	1000	-
w_{col}	1	-	w_{as}	1	-
s_{margin}	10	m	w_{ad}	10	-
w_v	2	-	w_{js}	1	-
w_A	0.1	-	w_{jd}	10	-
T_{st}	3	s	A	[-2:0.5:1]	m/s^2

A. TRAFFIC GENERATION

The obstacle vehicles are controlled by an IDM. To imitate human drivers who might be aggressive or defensive, both types of vehicles are included in the environment for the first two scenarios. Only one type of vehicle was considered for test scenarios 3 and 4. The dynamics of the vehicle are described by the following equations:

$$\dot{x} = v \quad (36)$$

$$\dot{v} = a\left(1 - \left(\frac{v}{v_0}\right)^4 - \left(\frac{s^*(v, \Delta v)^2}{s}\right)\right) \quad (37)$$

where $s^*(v, \Delta v) = s_0 + vT + v\Delta v/2\sqrt{ab}$, v_0, s_0, T, a and b are the model parameters. The parameters of each vehicle are listed in Table 3.

B. SCENARIOS

1) SCENARIO 1 (FAST DYNAMIC OBSTACLE SCENARIO)

In this scenario, dynamic obstacles are set to have relatively high-speed profiles using the IDM parameters as shown in Table 3. A total of 10 dynamic obstacles were arbitrarily positioned on two-lane roads within a certain distance in front of the ego vehicle. The ego vehicle is expected to perform appropriate lane change and car-following maneuvers, generating a safe and smooth spatiotemporal trajectory.

TABLE 3. IDM parameters used for different driving styles in test scenarios.

Parameters	v_0	s_0	a	b	T
unit	m/s	m	m/s^2	m/s^2	s
Scen.1 aggressive driver	20	12	5	5	0.4
Scen.1 defensive driver	18	25	2	2	1.3
Scen.2 aggressive driver	10	12	5	5	0.4
Scen.2 defensive driver	10	25	2	2	1.3
Scen.3	0	12	5	5	0.4
Scen.4	20	12	5	5	0.4

In Fig. 10(a), the upper figure shows the results of the motion-planning algorithm. The *long-term* path (blue) is the output of the first part of the algorithm, and the *short-term* trajectory (light blue) is the result of the last part of the algorithm. The bottom left figure shows the result of the JPS algorithm when solving the problem of the second part of the algorithm in the distance-time plane by displaying the planned trajectory and jump point. The bottom-right figure shows the velocity profile generated by the algorithm. The reference velocity was set to 20 m/s. The *long-term* profile (blue) is from the second part of the algorithm, and the *short-term* profile (light blue) is from the last part of the algorithm. In Fig. 10(a), the ego vehicle undergoes a lane change while accelerating to the reference speed. Next, in Fig. 10(b), the ego vehicle keeps the lane while maintaining the reference speed. Finally, in Fig. 10(c), the ego vehicle slows down while performing a car-following maneuver. There are two obstacles in the right lane, therefore a lane maintenance path is generated for the *long-term* path. Finally, the results in Fig. 10(d) show both the reference velocity and velocity profile of the ego vehicle during the entire scenario. In Fig. 11, the plots show the trajectories of the ego vehicle and obstacles within a certain time gap. The blue trajectory depicted with a triangle inside belongs to the ego vehicle, and the other trajectories belong to the obstacles.

2) SCENARIO 2 (SLOW DYNAMIC OBSTACLE SCENARIO)

In this scenario, dynamic obstacles are set to have relatively low-speed profiles, as shown in Table 3. This scenario aims to evaluate the lane change and car-following performance in low-speed traffic. In Fig. 12(a), a *long-term* path is generated to change the lane considering the surrounding obstacles. As shown in Fig. 12(b), car-following is performed because the risk of a lane change is high owing to the obstacle behind in the right lane. Then in Fig. 12(c), the ego vehicle performs a safe lane change maneuver as there is sufficient space. In the JPS plot in Fig. 12(c), while changing the lane and avoiding collision with obstacles, the planned trajectory shows that the ego vehicle tries to accelerate to the reference velocity. The overall maneuvering is shown in Fig. 13.

3) SCENARIO 3 (STATIC OBSTACLE SCENARIO)

In this scenario, static obstacles are placed in front of the ego vehicle. This scenario aims to evaluate the obstacle

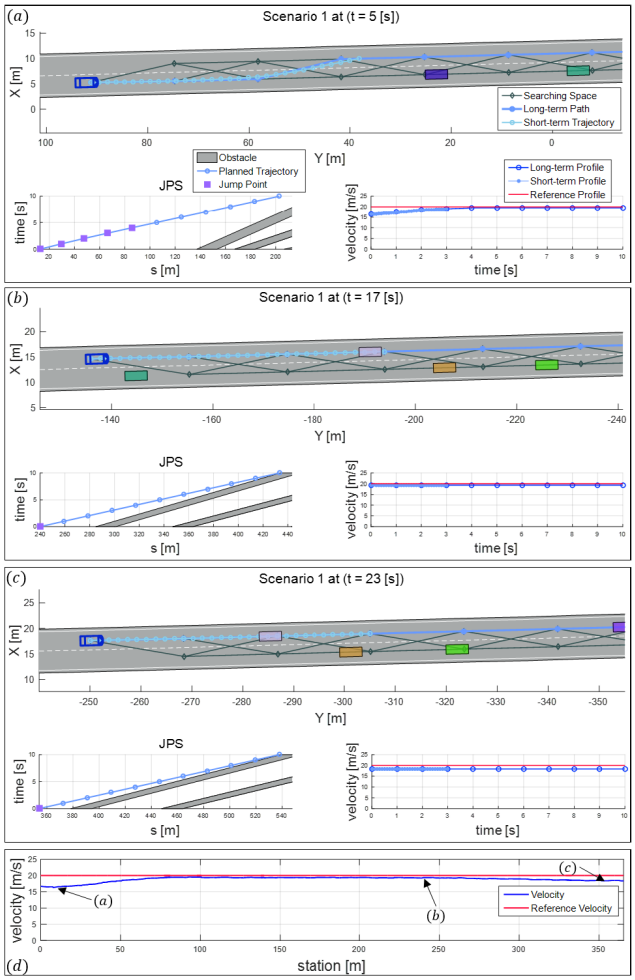

FIGURE 10. Evaluation in scenario 1: The ego vehicle (blue) (a) chooses a lane change maneuver; (b) keeps speed to the reference velocity; (c) decreases speed while performing a car-following maneuver. (d) Speed profile of the ego vehicle during the scenario.

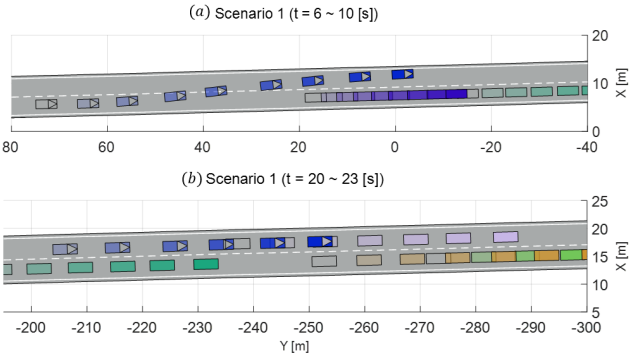

FIGURE 11. Trajectories of the ego vehicle and obstacles in scenario 1.

avoidance and stopping performance of the proposed algorithm. In Fig. 14(a), the ego vehicle avoids collision with a static obstacle ahead by choosing a *long-term* path for lane changing. As can be seen in the JPS plot, an optimal *long-term* trajectory is obtained which decelerates through

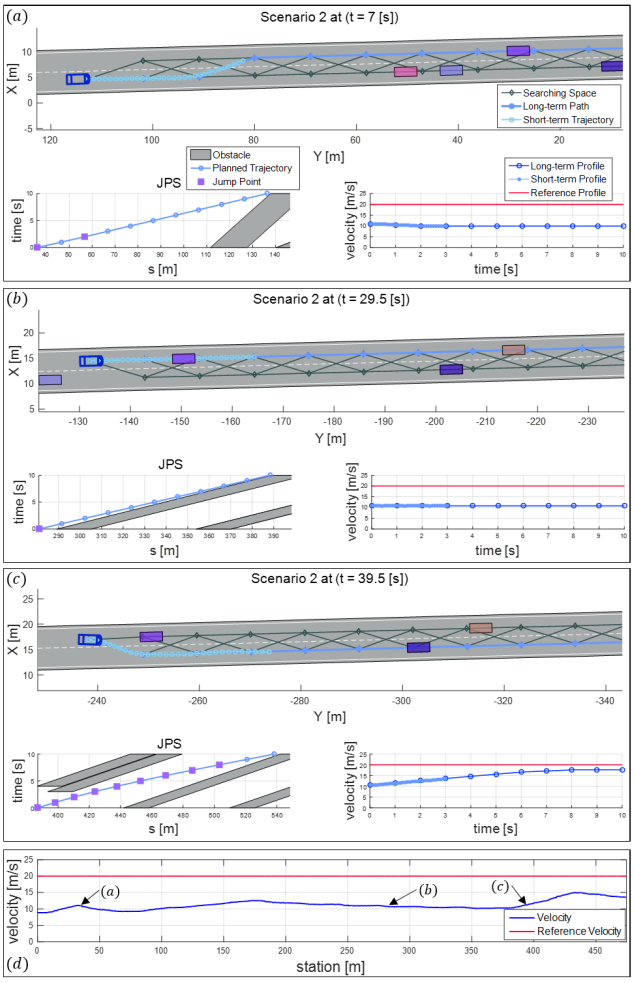

FIGURE 12. Evaluation in scenario 2: The ego vehicle (blue) (a) chooses a lane change maneuver; (b) performs a car-following maneuver; (c) performs another lane change when there is enough space. (d) Speed profile of the ego vehicle during the scenario.

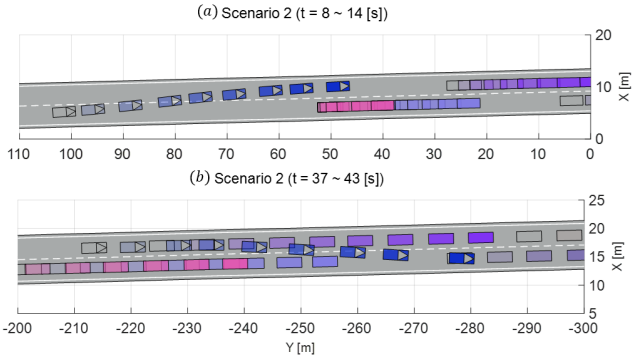

FIGURE 13. Trajectories of the ego vehicle and obstacles in scenario 2.

the optimal *long-term* path while keeping it collision-free. In Fig. 14(b) and (c), it can be seen that the ego vehicle continuously decelerates and finally stops to avoid collision with the obstacles. The overall maneuvering is shown in Fig. 15.

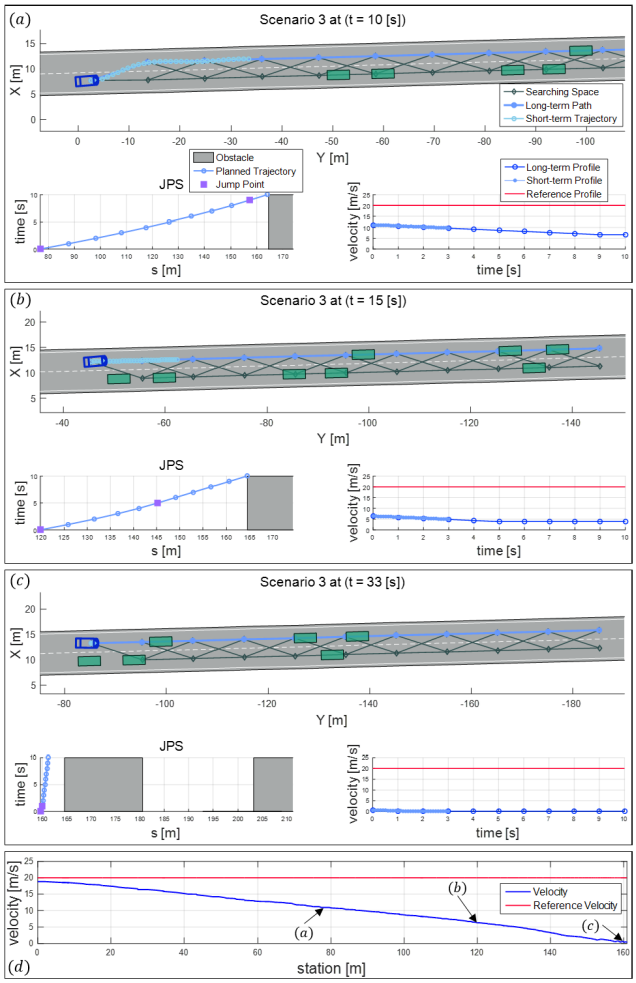

FIGURE 14. Evaluation in scenario 3: The ego vehicle (blue) (a) performs a lane change maneuver; (b) decreases speed smoothly; (c) stops behind the static obstacles. (d) Speed profile of the ego vehicle during the scenario.

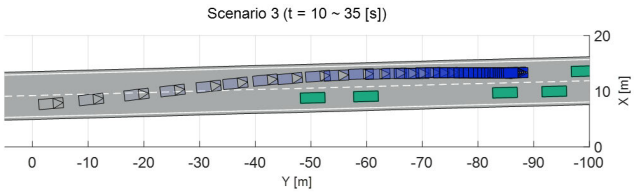

FIGURE 15. The trajectory of the ego vehicle in scenario 3.

4) SCENARIO 4 (HAZARD DYNAMIC OBSTACLE SCENARIO)

In this scenario, the hazard dynamic obstacles overlap with the lane. As shown in Fig. 16(a), the ego vehicle first changes lanes. Then, then, the *long-term* trajectory is generated as if there are no obstacles, as shown in Fig. 16(b). However, the *short-term* trajectory is generated considering the obstacles with collision cost described in Section IV-C, resulting in a trajectory that deviates from the reference trajectory. Next, as shown in Fig. 16(c), a *short-term* trajectory is generated

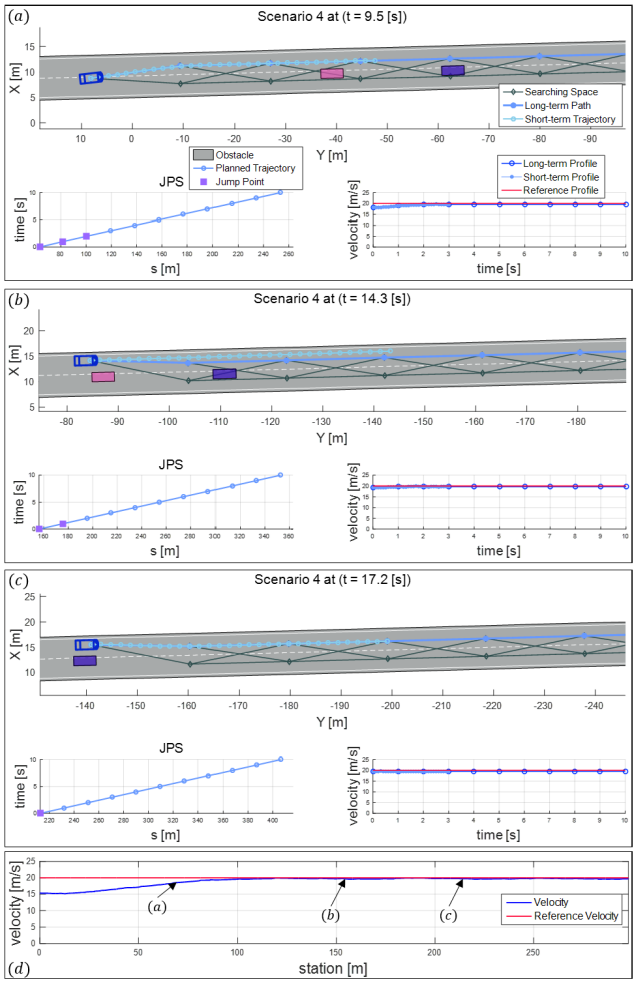

FIGURE 16. Evaluation in scenario 4: The ego vehicle (blue) (a) chooses a lane change maneuver; (b) performs an obstacle avoidance maneuver to avoid a collision with the surrounding vehicles; (c) plans a *short-term* trajectory toward the centerline when there is no more collision risk. (d) Speed profile of the ego vehicle during the scenario.

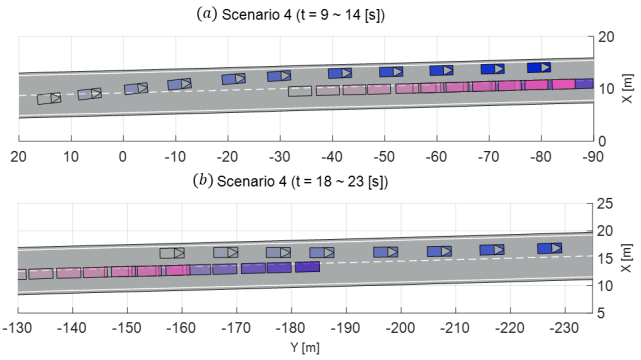

FIGURE 17. The trajectory of the ego vehicle in scenario 4.

toward the centerline when the collision risk disappears. The overall maneuvering is shown in Fig. 17.

5) SCENARIO 5 (CUT-IN SCENARIO)

In this last scenario, a dynamic obstacle performs an abrupt cut-in maneuver in terms of duration and relative speed [44].

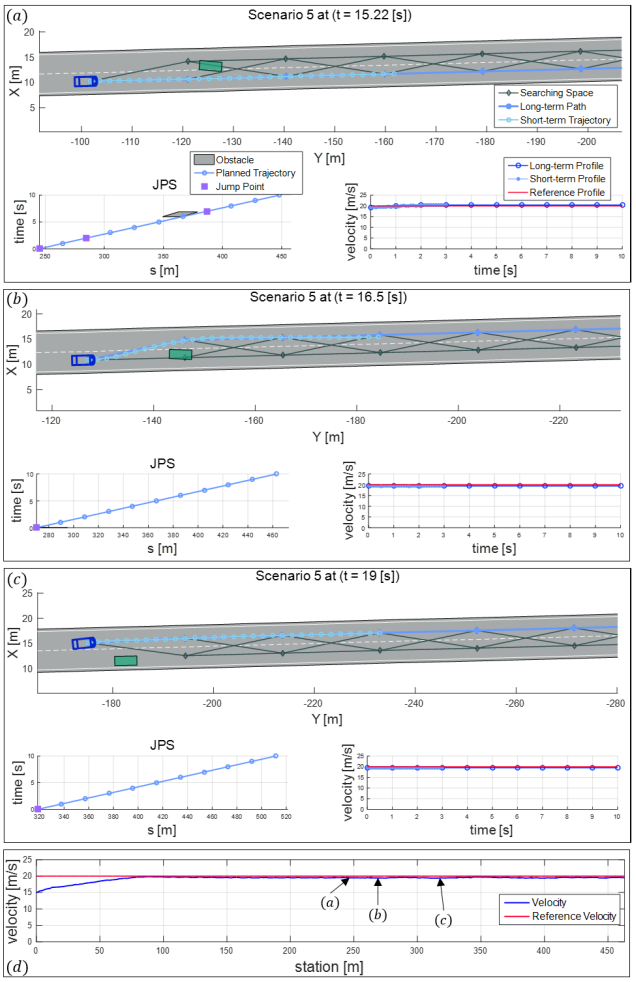

FIGURE 18. Evaluation in scenario 5: The ego vehicle (blue) (a) keeps the lane; (b) performs a lane change maneuver due to the cut-in vehicle ahead; (c) successfully performed a lane change. (d) Speed profile of the ego vehicle during the scenario.

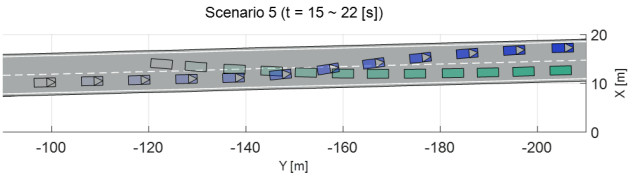

FIGURE 19. The trajectory of the ego vehicle in scenario 5.

The obstacle performs a lane change for 3 s, the relative speed with the ego vehicle is -5 m/s, and the lane change starts 25 m ahead. This scenario is intended to see how the planner reacts to a lane change of the obstacle. In Fig. 18(a), the ego vehicle maintains the lane while avoiding collision, as shown in the JPS plot. Then, in Fig. 18(b), the ego vehicle avoids collision with the cut-in obstacle ahead by choosing a *long-term* path for lane changing. The ego vehicle successfully performs a lane change to the left, as shown in Fig. 18(c). The overall maneuvering is illustrated in Fig. 19.

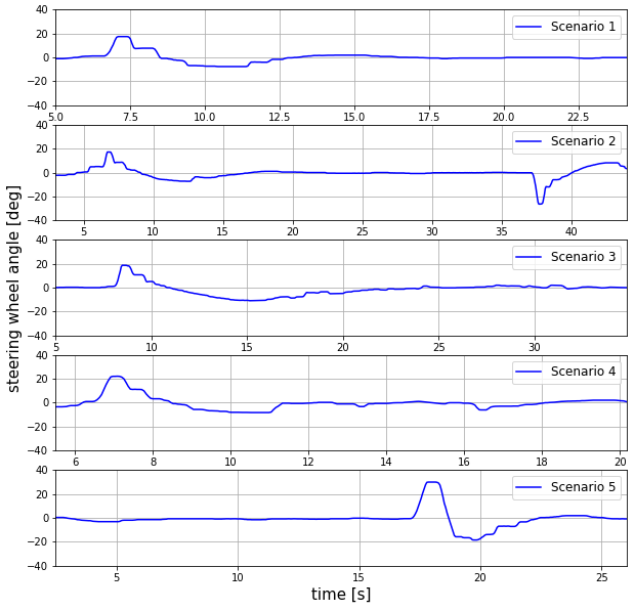

FIGURE 20. The steering wheel angle of the ego vehicle in all test scenarios.

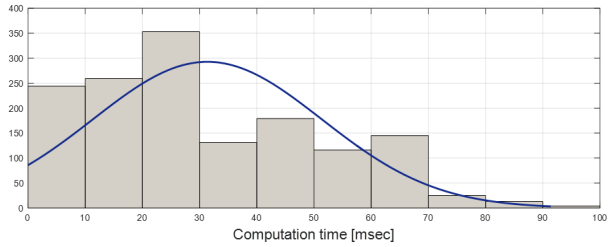

FIGURE 21. Histogram of the computation time for all scenarios.

The steering wheel angle of the ego vehicle for all test scenarios is shown in Fig. 20. The results show that appropriate steering wheel angle is applied to the ego vehicle in order to perform lane change maneuvers. In the test scenario 5, it is seen that an abrupt lane change is performed by applying a larger steering wheel angle than other scenarios to avoid collision with the cut-in obstacle.

Remark 1: The selected five scenarios include representative situations that may be encountered while driving on a highway. There are static and dynamic obstacles, including hazard vehicles that deviate from the center of the lane. The autonomous vehicle performs lane keeping, following, lane changing, stopping and swerve for safe highway driving.

Remark 2: In the test scenarios, the *long-term* path planner provides a desired maneuver with a lane-level lateral targets considering the lanes and obstacles. The paths generated show appropriate planning strategy, especially in lane selection. The *long-term* trajectory planner then applies the position and velocity profiles to the optimal path to avoid collisions with obstacles determined to overlap the path. Experimental results show satisfactory performance covering all scenarios.

Remark 3: The last *short-term* trajectory planner optimizes the trajectory considering comfort and safety. The optimization using the particles of steering angle sets with an adequate vehicle model makes smooth and feasible trajectory. In addition, the obstacles not considered in *long-term* planner are considered in optimization procedure to make collision free trajectory.

Fig. 21 shows a histogram of the computation time of the proposed algorithm for all scenarios. The distribution of the computation time is expressed through a histogram with an additional normal distribution. The average computation time was 31 ms, and the worst-case computation time was 94 ms.

VI. CONCLUSION AND FUTURE WORK

In this paper, the proposed hierarchical motion planning framework was shown to be a safe and efficient solution for autonomous driving in structured environments with various static and dynamic obstacles. The proposal was evaluated by means of a VILS with a real car under five representative scenarios.

The *long-term* planner, which consists of two processes, first provides a rough path for the driving strategy over a long time horizon. Then, the JPS is utilized to reduce the computational burden of A*, giving an acceptable quality solution. The generated trajectory was safe, comfortable, and dynamically feasible. This long horizon planner gives a long-sighted solution which can handle various maneuvers in decision-making where a plenty of obstacles exist.

Next, the *short-term* planner optimizes the front part of the rough trajectory by first resampling and then applying the PSO. This procedure provides kinodynamically feasible trajectory, enhancing the quality of trajectory in terms of comfort, smoothness and safety. The performance is evaluated under five scenarios, which include virtual static and dynamic obstacles. The proposed method provides a feasible and reasonable path and trajectory for performing lane changing, car-following, obstacle avoidance, and stopping in an appropriate manner. Furthermore, the computation time of the algorithm was within 100 ms.

As future work, the proposed planning framework will be tested on a broader range of traffic scenarios including urban situations, to further verify and evaluate the performance. The traffic light scenario can be conducted where a long horizon capable planning is important in maneuver decision-making (to pass or to stop at traffic light). In addition, to meet the requirements of advanced prediction methods [45]–[48], interaction-aware trajectory prediction with uncertainty could be considered in motion planning instead of simple physical model-based predictions. Finally, it could be interesting to focus on applying the deep learning technique to make human-like behavior decision-making.

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] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 4, pp. 1135–1145, Nov. 2016.

[3] V. Beanland, M. Fitzharris, K. L. Young, and M. G. Lenné, "Driver inattention and driver distraction in serious casualty crashes: Data from the Australian national crash in-depth study," *Accident Anal. Prevention*, vol. 54, pp. 99–107, May 2013.

[4] *2015 Motor Vehicle Crashes: Overview*, Traffic Safety Facts: Research Note, National Highway Traffic Safety Administration, Washington, DC, USA, 2016, pp. 1–9.

[5] Y. Meng, Y. Wu, Q. Gu, and L. Liu, "A decoupled trajectory planning framework based on the integration of lattice searching and convex optimization," *IEEE Access*, vol. 7, pp. 130530–130551, 2019.

[6] J. Ziegler, P. Bender, T. Dang, and C. Stiller, "Trajectory planning for Bertha—A local, continuous method," in *Proc. IEEE Intell. Vehicles Symp.*, Jun. 2014, pp. 450–457.

[7] Z. W. Geem, *Recent Advances in Harmony Search Algorithm*, vol. 270. Cham, Switzerland: Springer, 2010.

[8] J. Chen, W. Zhan, and M. Tomizuka, "Autonomous driving motion planning with constrained iterative LQR," *IEEE Trans. Intell. Veh.*, vol. 4, no. 2, pp. 244–254, Jun. 2019.

[9] J. Ziegler and C. Stiller, "Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2009, pp. 1879–1884.

[10] M. McNaughton, C. Urmson, J. M. Dolan, and J.-W. Lee, "Motion planning for autonomous driving with a conformal spatiotemporal lattice," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2011, pp. 4889–4895.

[11] S. V. Dorff, M. Kneissl, and M. Franzle, "Safe, deterministic trajectory planning for unstructured and partially occluded environments," in *Proc. IEEE Int. Intell. Transp. Syst. Conf. (ITSC)*, Sep. 2021, pp. 969–975.

[12] C. Liu, S. Lee, S. Varnhagen, and H. E. Tseng, "Path planning for autonomous vehicles using model predictive control," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2017, pp. 174–179.

[13] D. Madas, M. Nosratinia, M. Keshavarz, P. Sundstrom, R. Philippsen, A. Eidehall, and K.-M. Dahlen, "On path planning methods for automotive collision avoidance," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2013, pp. 931–937.

[14] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a Frenét frame," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2010, pp. 987–993.

[15] X. Li, Z. Sun, D. Cao, Z. He, and Q. Zhu, "Real-time trajectory planning for autonomous urban driving: Framework, algorithms, and verifications," *IEEE/ASME Trans. Mechatronics*, vol. 21, no. 2, pp. 740–753, Apr. 2016.

[16] X. Li, Z. Sun, D. Cao, D. Liu, and H. He, "Development of a new integrated local trajectory planning and tracking control framework for autonomous ground vehicles," *Mech. Syst. Signal Process.*, vol. 87, pp. 118–137, Mar. 2017.

[17] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu Apollo EM motion planner," 2018, *arXiv:1807.08048*.

[18] W. Xu, J. Wei, J. M. Dolan, H. Zhao, and H. Zha, "A real-time motion planner with trajectory optimization for autonomous vehicles," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2012, pp. 2061–2067.

[19] Y. Zhang, H. Sun, J. Zhou, J. Pan, J. Hu, and J. Miao, "Optimal vehicle path planning using quadratic optimization for Baidu Apollo open platform," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Oct. 2020, pp. 978–984.

[20] H. Li, G. Yu, B. Zhou, P. Chen, Y. Liao, and D. Li, "Semantic-level maneuver sampling and trajectory planning for on-road autonomous driving in dynamic scenarios," *IEEE Trans. Veh. Technol.*, vol. 70, no. 2, pp. 1122–1134, Feb. 2021.

[21] Y. Zhang, H. Chen, S. L. Waslander, J. Gong, G. Xiong, T. Yang, and K. Liu, "Hybrid trajectory planning for autonomous driving in highly constrained environments," *IEEE Access*, vol. 6, pp. 32800–32819, 2018.

[22] W. Lim, S. Lee, M. Sunwoo, and K. Jo, "Hierarchical trajectory planning of an autonomous car based on the integration of a sampling and an optimization method," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 2, pp. 613–626, Feb. 2018.

[23] X. Jin, Z. Yan, G. Yin, S. Li, and C. Wei, "An adaptive motion planning technique for on-road autonomous driving," *IEEE Access*, vol. 9, pp. 2655–2664, 2021.

[24] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller, and E. Kaus, "Making Bertha drive—An autonomous journey on a historic route," *IEEE Intell. Transp. Syst. Mag.*, vol. 6, no. 2, pp. 8–20, 2014.

[25] C. Hubmann, M. Aeberhard, and C. Stiller, "A generic driving strategy for urban environments," in *Proc. IEEE 19th Int. Conf. Intell. Transp. Syst. (ITSC)*, Nov. 2016, pp. 1010–1016.

[26] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, and M. Gittleman, "Autonomous driving in urban environments: Boss and the urban challenge," *J. Field Robot.* vol. 25, no. 8, pp. 425–466, Aug. 2008.

[27] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, and O. Koch, "A perception-driven autonomous urban vehicle," *J. Field Robot.*, vol. 25, no. 10, pp. 727–774, 2008.

[28] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, and D. Johnston, "Junior: The Stanford entry in the urban challenge," *J. Field Robot.*, vol. 25, no. 9, pp. 569–597, 2008.

[29] M. Olsson, "Behavior trees for decision-making in autonomous driving," Tech. Rep., 2016.

[30] H. Kim, D. Kim, and K. Huh, "Intention aware motion planning with model predictive control in highway merge scenario," SAE, Warrendale, PA, USA, Tech. Rep., 2019.

[31] C. Hubmann, "Belief state planning for autonomous driving: Planning with interaction, uncertain prediction and uncertain perception," Tech. Rep., 2020.

[32] D. D. Harabor and A. Grastien, "Online graph pruning for pathfinding on grid maps," in *Proc. 25th AAAI Conf. Artif. Intell.*, 2011, pp. 1–6.

[33] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[34] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Proc. Congr. Evol. Comput. (CEC)*, vol. 3, Jan. 2003, pp. 1945–1950.

[35] N. SinghPal and S. Sharma, "Robot path planning using swarm intelligence: A survey," *Int. J. Comput. Appl.*, vol. 83, no. 12, pp. 5–12, Dec. 2013.

[36] H. I. Kang, B. Lee, and K. Kim, "Path planning algorithm using the particle swarm optimization and the improved Dijkstra algorithm," in *Proc. Pacific–Asia Workshop Comput. Intell. Ind. Appl. (PACII)*, vol. 2, Dec. 2008, pp. 1002–1004.

[37] Y. Guo, W. Wang, and S. Wu, "Research on robot path planning based on fuzzy neural network and particle swarm optimization," in *Proc. 29th Chin. Control Decis. Conf. (CCDC)*, May 2017, pp. 2146–2150.

[38] S. Thabit and A. Mohades, "Multi-robot path planning based on multi-objective particle swarm optimization," *IEEE Access*, vol. 7, pp. 2138–2147, 2018.

[39] U. Schwesinger, M. Rufli, P. Furgale, and R. Siegwart, "A sampling-based partial motion planning framework for system-compliant navigation along a reference path," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2013, pp. 391–396.

[40] B. Li, Y. Zhang, and Z. Shao, "Spatio-temporal decomposition: A knowledge-based initialization strategy for parallel parking motion optimization," *Knowl.-Based Syst.*, vol. 107, pp. 179–196, Sep. 2016.

[41] H. Pacejka, *Tire and Vehicle Dynamics*. Amsterdam, The Netherlands: Elsevier, 2005.

[42] M. Butenuth, R. Kallweit, and P. Prescher, "Vehicle-in-the-loop real-world vehicle tests combined with virtual scenarios," *ATZ Worldwide*, vol. 119, no. 9, pp. 52–55, Sep. 2017.

[43] X. Che, C. Li, and Z. Zhang, "An open vehicle-in-the-loop test method for autonomous vehicle," EasyChair, Tech. Rep., 2019.

[44] T. Toledo and D. Zohar, "Modeling duration of lane changes," *Transp. Res. Rec., J. Transp. Res. Board*, vol. 1999, no. 1, pp. 71–78, Jan. 2007.

[45] H. Ma, J. Li, W. Zhan, and M. Tomizuka, "Wasserstein generative learning with kinematic constraints for probabilistic interactive driving behavior prediction," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2019, pp. 2477–2483.

[46] C. Dong, Y. Chen, and J. M. Dolan, "Interactive trajectory prediction for autonomous driving via recurrent meta induction neural network," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 1212–1217.
[47] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, "Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data," 2020, *arXiv:2001.03093*.
[48] H. Kim, D. Kim, G. Kim, J. Cho, and K. Huh, "Multi-head attention based probabilistic vehicle trajectory prediction," 2020, *arXiv:2004.03842*.

DONGCHAN KIM received the B.S. degree in automotive engineering from Hanyang University, Seoul, Republic of Korea, in 2015. He is currently pursuing the Ph.D. degree with the Department of Automotive Engineering. His research interests include motion planning, trajectory prediction, vehicle state estimation, reinforcement learning, and deep learning applications to autonomous vehicle.

GIHOON KIM received the B.S. degree in automotive engineering from Hanyang University, Seoul, Republic of Korea, in 2017. He is currently pursuing the Ph.D. degree with the Department of Automotive Engineering. His research interests include interaction-aware driving, which is planning the path considering effects to other vehicles from ego vehicle's movement and development of control system for collision avoidance using aggressive-maneuver driving like drift control.

HAYOUNG KIM received the B.S. degree in mechanical engineering from Hanyang University, Seoul, Republic of Korea, in 2014, where he is currently pursuing the Ph.D. degree with the Department of Automotive Engineering. His research interests include interaction-aware motion planning, predictions, deep learning, and model predictive control for automated vehicle.

KUNSOO HUH (Member, IEEE) received the Ph.D. degree from the University of Michigan, Ann Arbor, MI, USA, in 1992. He is currently a Professor with the Department of Automotive Engineering, Hanyang University, Seoul, South Korea. His research interests include machine monitoring and control, with emphasis on their applications to vehicular systems, sensor-based active safety systems, V2X-based safety systems, autonomous vehicle control, and AI applications in autonomous vehicle.