# Recursive Path Planning Using Reduced States for Car-Like Vehicles on Grid Maps

Sangyol Yoon, Sung-Eui Yoon, *Senior Member, IEEE*, Unghui Lee, and David Hyunchul Shim

***Abstract*—We present a recursive path-planning method that efficiently generates a path by using reduced states of the search space and taking into account the kinematics, shape, and turning space of a car-like vehicle. Our method is based on a kinematics-aware node expansion method that checks for collisions based on the shape and turning space of a vehicle. We present two heuristics that simultaneously consider the kinematics of a vehicle with and without obstacles. In particular, for challenging environments containing complex obstacles and even narrow passages, we recursively identify intermediate goals and nodes that allow the vehicle to compute a path to its destination. We show the benefits of our method through simulations and experimental results by using an autonomous ground vehicle. Furthermore, we show that our method can efficiently generate a collision-free path for vehicles in complex environments with passageways.**

***Index Terms*—A$^*$, path planning, car-like vehicle, turning space.**

## I. Introduction

Autonomous vehicles have lately drawn considerable attention, especially following the successes of the Defense Advanced Research Projects Agency's (DARPA) Grand and Urban Challenges [1], [2] and Google's Self-Driving Car [3]. Autonomous vehicles are built atop various functional blocks, including sensing technologies and other controls. At a high level of abstraction, autonomous driving consists of four steps [4]: 1) perceiving the environment of the vehicle [5], [6], 2) localizing the vehicle in the environment [7], [8], 3) generating a collision-free path to the chosen goal [9], [10], and 4) operating the vehicle as desired to follow the path [11], [12]. In this paper, we focus on the efficient generation of collision-free paths that vehicles can easily follow in various environments.

Considerable research has been devoted in the last few decades to generating collision-free paths for autonomous ground vehicles [10]. Most existing approaches can be classified into grid- and sampling-based path planners [13]. In the grid-based approach, the A$^*$ algorithm is known to be very effective at finding the shortest path to goal while avoiding obstacles [14], [15]. Many variants that purport to overcome the drawbacks associated with A$^*$ have been developed. Notable approaches include identifying fewer grid edges (e.g., Field D$^*$ [16] and Theta$^*$ [17]), designing better heuristics by considering the kinematics of the vehicle or the obstacles [18]–[20], respecting the shapes of vehicles [21], replanning (e.g., Lifelong Planning A$^*$ (LPA$^*$) [22] and D$^*$ Lite [23]), post-processing [24], etc. Sampling-based approaches include the Probabilistic Road Map (PRM) [25], [26], Rapidly-exploring Random Trees (RRT) [27], and RRT$^*$ [28].

Computing an optimal, collision-free path for car-like vehicles with nonholonomic constraints has been known to be NP-hard [29]. As mentioned above, several approaches have been proposed to improve both grid- (variants of A$^*$) and sampling-based (variants of PRM and RRT) algorithms for car-like vehicles. Grid-based techniques have been adopted for autonomous vehicles and have shown successful results [20]. This is mainly because grid-based techniques tend to be more efficient than sampling-based path planners, especially when the dimensions of the state spaces are fewer than six [13]. One can also use RRT$^*$, but it generally runs on the order of a few seconds, although its efficiency depends on the size of the map of the path in question [30]. Furthermore, if no collision-free path to goal exists, grid-based path planners can report this failure much more quickly than sampling-based ones.

In light of this, we focus on improving existing grid-based path planners that efficiently compute collision-free paths in a discrete space for a given environment with obstacles. Existing grid-based path planners are not designed for complex environments containing many obstacles or narrow passages, and thus tend to either run slowly or fail to find paths to goal. In this paper, we present an efficient, recursive path-planning method operating with grid maps and two online heuristic functions that takes into account the kinematics, shape, and turning space of car-like vehicles and uses reduced states of the search space to compute the shortest path to goal. The main idea underlying our method is that when we cannot find a path in an initial attempt in complex scenarios, we identify intermediate goals and nodes and recursively find paths to these goals and nodes. We use simulations and conduct experiments with an autonomous vehicle to show the benefits of our method in challenging scenarios (Fig. 1).

## II. Related Works

In this section, we discuss past research directly related to our work.

Manuscript received September 16, 2014; revised December 10, 2014 and February 12, 2015; accepted April 9, 2015. Date of publication May 4, 2015; date of current version September 25, 2015. This work was supported by the National Research Foundation of Korea, funded by the Korean Government (MEST), under Grant 20110031920. The Associate Editor for this paper was W. Fan.

S. Yoon is with LG Electronics Inc., Incheon 404-170, Korea (e-mail: sangyol.yoon@kaist.ac.kr).

S.-E. Yoon is with the Department of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon 305-701, Korea (e-mail: sungeui@gmail.com).

U. Lee and D. H. Shim are with the Department of Aerospace Engineering, Korea Advanced Institute of Science and Technology, Daejeon 305-701, Korea (e-mail: lamer0712@kaist.ac.kr; hcshim@kaist.ac.kr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITS.2015.2422991

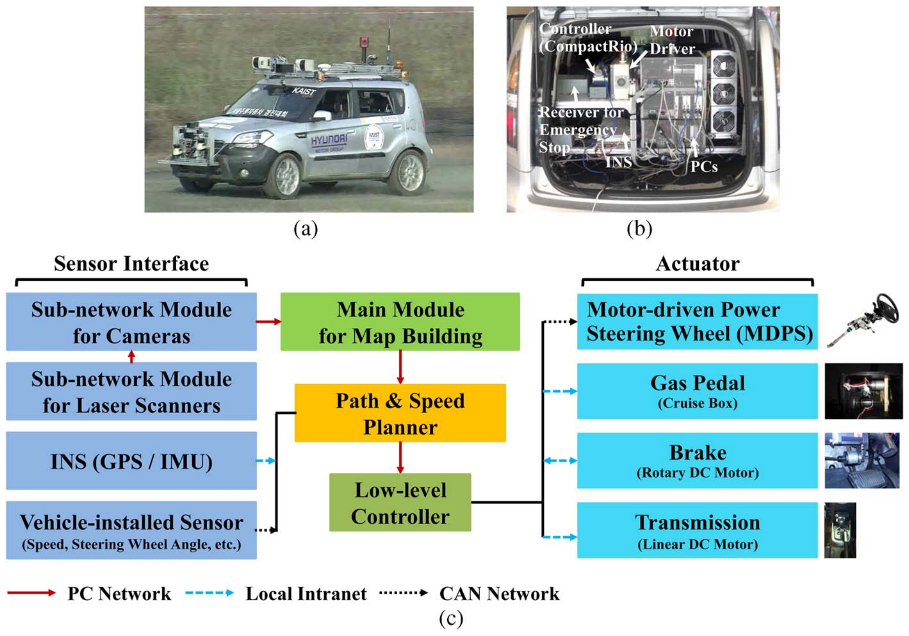

Controller (CompactRio)
Motor Driver
Receiver for Emergency Stop
INS
PCs
(a)
(b)
Sensor Interface
Sub-network Module for Cameras
Sub-network Module for Laser Scanners
INS (GPS / IMU)
Vehicle-installed Sensor (Speed, Steering Wheel Angle, etc.)
Main Module for Map Building
Path & Speed Planner
Low-level Controller
Actuator
Motor-driven Power Steering Wheel (MDPS)
Gas Pedal (Cruise Box)
Brake (Rotary DC Motor)
Transmission (Linear DC Motor)
PC Network
Local Intranet
CAN Network
(c)

Fig. 1. (a) KAIST autonomous ground vehicle, (b) built-in computer systems for path planning, inertial navigation system (INS) operation, steering wheel control, etc., and (c) overall system architecture.

## A. A* Algorithms

A* algorithms have been extensively studied and are known to be effective for finding the shortest path to goal using grid maps [14], [15]. A brief review of these algorithms is provided in Section III. Earlier techniques of this sort tended to find zigzag paths when the goal was not located on a horizontal, a vertical, or a 45-° slope to the starting position. Moreover, most techniques perform their procedures without using information from nodes already searched, even when the obstacles move slightly.

To overcome these problems, techniques that identify fewer grid edges (Field D* [16] and Theta* [17]) and replanning by using information from nodes already searched (LPA* [22] and D* Lite [23]) have been proposed. However, these techniques do not take into account the kinematics, shape, and turning space of car-like vehicles.

## B. Heuristic

In general, Euclidean distance is used as the heuristic for several types of A* algorithms. However, this often results in the algorithm performing poorly because the quality of the heuristic function affects the search time for goal [31].

Ziegler *et al.* [18], Likhachev *et al.* [19], and Dolgov *et al.* [20] proposed methods that use two kinds of complementing heuristics such that the one yielding the maximum value is chosen as the final heuristic. The first heuristic takes into account the kinematics of the car-like vehicle by assuming an obstacle-free environment, whereas the second one considers obstacles but ignores the vehicle's kinematics. The first heuristic is based on rotation-translation-rotation paths. Alternatively, it can first make offline calculations given precomputed parameters, including the initial and target positions and orientations of the vehicle, and then translate and rotate to account for configurations that are not precomputed. On the other hand, the second heuristic is computed online using a Voronoi graph or dynamic programming regardless of the kinematics of the car-like vehicle. The above-mentioned researchers showed that a combination of the two heuristics effectively reduces the search time for the path to goal. However, these complementing heuristics cannot simultaneously accommodate kinematics, thus resulting in expansions to unpromising regions. Moreover, the second heuristic can require longer to compute a solution because of the Voronoi graph or dynamic programming used.

Our proposed method uses the two types of heuristics described above. Unlike prior approaches, however, both our online obstacle-free and obstacle-aware kinematic heuristics consider the kinematics of the car-like vehicle. Furthermore, our obstacle-aware kinematic heuristic can be efficiently evaluated online within the A* framework, since it is based on the geometric concept of the Dubins model [32]. Our obstacle-free kinematic heuristic computes the shortest distance from a current position to goal while considering the kinematics of the car-like vehicle by assuming an obstacle-free environment, i.e., it is more aggressive and focuses on identifying the shortest path to goal, whereas our obstacle-aware kinematic heuristic computes the obstacle distance while considering its kinematics in the presence of obstacles, i.e., it is more conservative and focuses on identifying paths that avoid obstacles.

## C. Narrow Passages

Computing a collision-free path in environments with many obstacles and narrow passages poses significant technical challenges for path planners. To address them, many variants of PRM [25], [26], RRT [27], and RRT* [28] have been proposed. At a high level of abstraction, the probability of finding a path through narrow passages depends upon the sampling density and the sampling strategies. A few prior techniques employed adaptive sampling [33], hybrid approaches using approximate cell decomposition [34], free-space information [35], and retraction techniques that utilize the boundary of the obstacle space [36]. Nonetheless, finding paths to goal by taking into account the kinematics of car-like vehicles has not been extensively researched for environments with narrow passages.

A few approaches for A$^*$ techniques have been proposed for finding a collision-free path in narrow passages. One proposed method [37] for PRM techniques used a kind of regular structure, i.e., an adaptive grid structure commonly generated by approximate cell decomposition, and captured the connectivity of free space between the cells of the regular structure in a manner used for roadmap construction. These techniques are not primarily designed for use in A$^*$ methods, but can be applied to A$^*$ techniques. However, such techniques do not consider the kinematics of car-like vehicles.

By contrast, our method directly handles the narrow passage problem within the A$^*$ framework. When our method cannot compute a collision-free path to goal in an initial attempt, we treat it as involving narrow passages, and recursively compute intermediate goals and nodes by utilizing expanded node information extracted from our prior attempts to compute a path to goal.

## *D. Shape and Turning Space of Vehicles*

Ground vehicles can be broadly classified into tank-like and car-like vehicles. A tank-like vehicle can rotate on a point, whereas a car-like vehicle driven by front or rear wheels, the focus of this paper, has nonholonomic constraints on its movement [38]. As a result, we need to consider its turning space as well as its shape.

In their research, Likhachev and Ferguson treated the vehicle as a point, and generated two kinds of obstacle maps by considering the inner and outer radii of the circle surrounding the vehicle [19]. This method tends to be expensive because we need to expand obstacles with the inner and outer radii. Additionally, it does not consider the turning space and thus cannot make a tight turn. Recently Dolgov *et al.* find a collision-free path with a largest clearance based on the center of the rear axle of the vehicle without considering the turning space [20]. It then smooths a collision-free path, while checking the turning space. This process is repeated until the computed path is confirmed to be collision-free. This approach is rather time-consuming due to the nature of iterative process.

On the other hand, our method represents a car-like vehicle as a rectangle instead of a circle, and directly considers the turning space, while expanding nodes and evaluating heuristics. As a result, our approach can efficiently identify a collision-free path considering the shape and turning space of the car-like vehicle.

## III. Overview

In this paper, we focus on a path planner that generates a collision-free path by considering the kinematics, shape, and turning space of car-like vehicles. We then assume that autonomous ground vehicles follow such paths by smoothing the paths for better driving conditions and using path-following methods. Many approaches have been developed for path following of this sort [4], [12], [39]. A path-following controller is responsible for tracking the generated path with minimal error.

TABLE I
Legend for Boxes and Triangles Used in the Grid Map. The Line Used in a Triangle Indicates the Orientation of the Vehicle. See the PDF File for Their Color Difference

| Type | Meaning |
|---|---|
| 🟩 | Start node |
| 🟨 | Goal node |
| ⬛ | Obstacle |
| 🟦 | Node in a discovered path |
| ▨ | Expanded node |
| ▷ | Node expanded with forward movement |
| ▷ | Node expanded with reverse movement |
| ▷ | Node expanded with orientation |

We will show examples and images of paths in a grid map representing obstacles. Throughout this paper, we use the legend for such plots described in Table I, unless otherwise indicated.

### *A. Notations*

Our work utilizes the conventional A$^*$ algorithm. In this subsection, we define terms used throughout this paper.

As in the common A$^*$ algorithm, a node be in one of the following statuses: "unvisited," "open," or "closed" [14]. When a node $n$ is expanded to its child node $n'$, the distance between them is associated with an *arc cost*. If the current node $n$ is arrived at through several expansions from the start node $n_s$, the sum of the arc costs of those expansions from the start node $n_s$ to the current node $n$ is defined as the *path cost*. Further, a conservatively estimated cost from the current node $n$ to the goal node $n_g$ is called the cost-to-go *heuristic*. In general, the heuristic used by conventional A$^*$ is the Euclidean distance. The sum of the path cost and the heuristic for a node is defined as the *evaluation cost*. When a heuristic is *consistent*, it is unnecessary to reopen nodes that are "closed" [40]. The consistency condition is defined as follows:

$$h(n) \leq c(n,\ n') + h(n') \qquad (1)$$

where $h(n)$ and $h(n')$ are the heuristics of the current node $n$ and its child node $n'$, respectively, and $c(n, n')$ is the arc cost from node $n$ to $n'$.

When the heuristic function satisfies the following three conditions and there exists a path to the goal, the A$^*$ algorithm is guaranteed to find the optimal path to goal [41]. These conditions are called *admissibility* conditions and defined as follows: 1) each node in a grid (or a graph) has a finite number of successors, 2) all arc costs are positive, and 3) the heuristic is conservative, i.e., for all nodes, the heuristic must never overestimate the actual value.

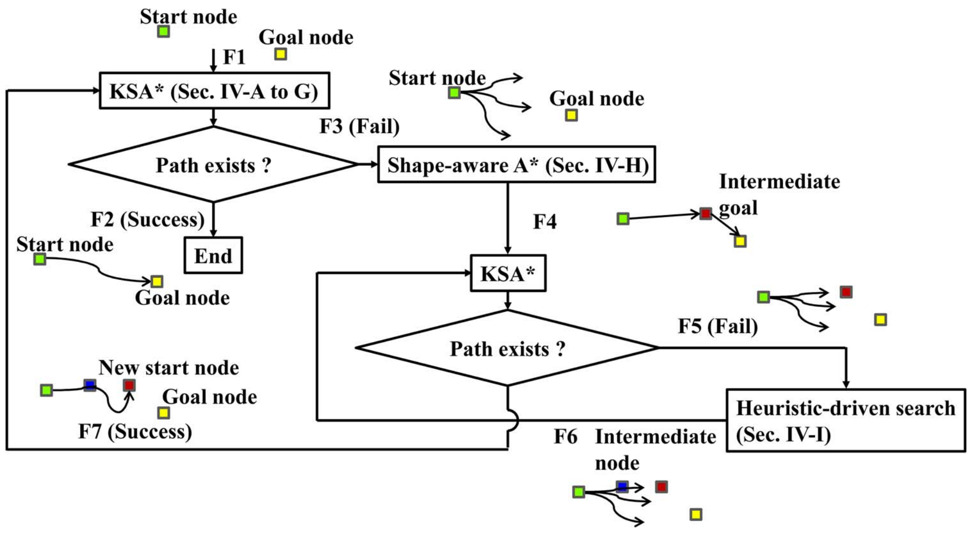


Fig. 2. This figure shows the overall flow of our method, with examples and the running sequence represented using numbers. We first run our KSA* algorithm given the start and goal positions (F1). If it finds a path (F2), we return the path to the path-following control module in the autonomous vehicle. Otherwise (F3), we then recursively attempt to find a path by identifying intermediate goals (F4) and nodes (F6).

## B. Overview of Our Approach

To accommodate the kinematics of car-like vehicles and find paths even for narrow passages, we propose our Kinematics- and Shape-aware A$^*$ (KSA$^*$) algorithm. We define the state of each node to represent information related to the kinematics of the vehicle, including a position, an orientation, and a forward/reverse direction. For the state of a search space in order to construct trees for each node, however, we consider only an x-y position among the available states for greater efficiency; other state information, such as orientation and direction, is derived from the position of the node relative to neighboring nodes. Moreover, we only consider x-y positions to check whether each node is duplicated in the search space. Fig. 2 shows the flow of our recursive path-planning algorithm, which is introduced in Sections IV-H and IV-I.

We propose using a kinematics-aware node expansion method (Section IV-A) to construct a collision-free path that the vehicle can follow at a minimum turning radius. Our node expansion method is based on our orientation-driven arc costs (Section IV-B) to provide a comfortable driving experience by regulating the steering action of the vehicle. These methods are combined with our online kinematic heuristics (Sections IV-C, IV-D and IV-E) that consider both obstacle-free and obstacle-ridden environments to effectively reduce search time. In order to generate more realistic collision-free paths, we take into account the turning space of the vehicle as well as its shape.

For complex environments with several obstacles or narrow passages, our KSA$^*$ algorithm consisting of the aforementioned components may not find a path. In this case, we recursively find a path by identifying intermediate goals or nodes using either our shape-aware A$^*$ or heuristic-driven search (Sections IV-H and IV-I). We finally smooth the computed path to provide a comfortable driving experience and better path-following performance (Section IV-G), and pass this information on to the path-following module.

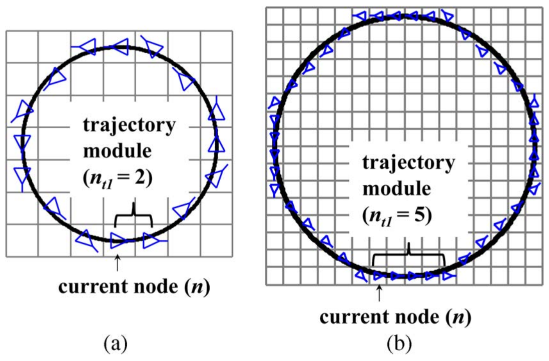

trajectory module ($n_{tl}=2$)
current node ($n$)
(a)
trajectory module ($n_{tl}=5$)
current node ($n$)
(b)

Fig. 3. Kinematics-aware node expansion. (a) Shows straight-edge patterns with $n_{t1}=2$, which approximates the minimum turning radius. (b) Shows node expansions with $n_{t1}=5$.

## IV. Proposed Algorithms

In this section, we explain each component of our method.

### A. Kinematics-Aware Node Expansion

It is critical to respect the kinematics of a car-like vehicle during node expansions in order to compute paths that can be easily taken by such vehicles. For this, we propose a kinematics-aware node expansion method. Our expansion method allows node expansion in three forward directions: straightforward, and left/right forward turns. We also allow three reversing directions corresponding to each forward expansion.

It is easy to expand nodes straight forward. To efficiently handle left and right turning cases, we discretize a circle of the minimum turning radius for the vehicle using grid edges. In particular, we approximate the circle by a series of straight lines consisting of $n_{t1}$ straight edges (*trajectory modules*, e.g., $n_{t1}=2$ and 5 in Fig. 3).

When the vehicle aims to turn left or right with its minimum turning radius $r_{\min}$, we allow an orientation change (e.g., following a diagonal edge from the horizontal edge) in the same

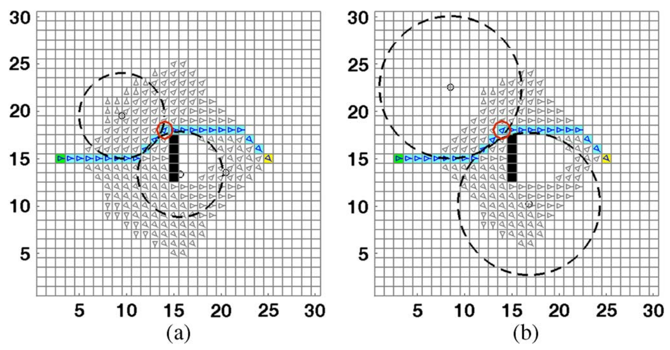


Fig. 4. These figures show that we can turn left while satisfying the minimum turning radius and can then turn right, irrespective of the conditions for orientation change (shown in the red circle). $r_{\min} = 4.5$ m, $n_{t1} = 3$. (b) $r_{\min} = 7.5$ m, $n_{t1} = 5$.

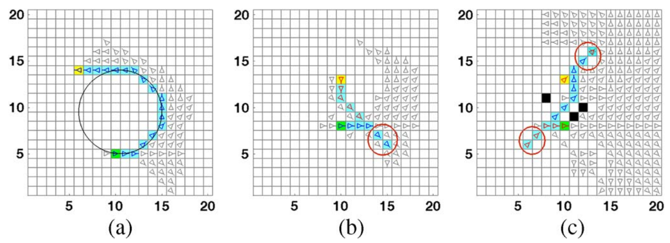


Fig. 5. Our method supports different turning maneuvers. The red circles represent the point of change in the direction of movement of the vehicle. The child nodes in red circles are only included when determining the final path by checking if there exist changes in the direction of movement in the final path to goal but not when expanding nodes. However, interference against obstacles is checked in the child nodes when expanding nodes. $r_{\min}$ is set to 4.5 m and $n_{t1}$ to 3. (a) U-turn. (b) One-point turn. (c) Two-point turn.

direction (forward/reverse) as the vehicle only after expanding $n_{t1}$ nodes with the same orientation and the same direction. To reverse the vehicle, we use a similar constraint. If the vehicle turns left, it is always allowed to turn right irrespective of the conditions for expanding $n_{t1}$ straight edges (Fig. 4), since this kind of turning always satisfies the minimum turning radius. The vehicle is also permitted to turn left immediately after it turns right.

Our simple kinematics-aware node expansion technique can easily support U-turns, one-point turns, and two-point turns (Fig. 5). When the goal is located inside the minimum turning radius, a one-point turn requires less space than a U-turn and is thus preferred. To check whether a one-point turn (in case of reverse movement following forward movement) is possible, interference against obstacles with regard to subsequent forward movement is gauged by turning half of $n_{t1}$ child nodes by 45° [the red circle in Fig. 5(b)]. For simply expanding nodes, however, the gauged child nodes are only generated in a final path when there are changes in the direction of movement in the final path, although the nodes are checked for interference against obstacles during node expansions. The circular arc created by such a 45° turning pattern is geometrically sufficient to accomplish such maneuvering. We compute a similar circular arc for the reverse movement.

The kinematics-aware node expansion described above is efficient, since we can perform numerous maneuvers based only on six grid edges. However, when the grid resolution is insufficient, our discretized trajectory module, given a minimum turning radius, can also realize a low resolution. In practice, we can use one meter or less as the width of each cell, by virtue of the efficiency of kinematics-aware node expansion.

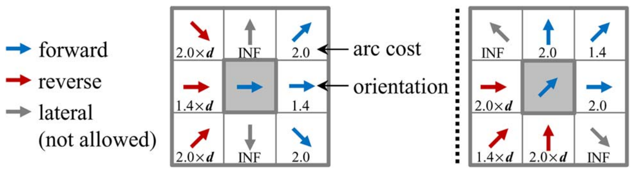


Fig. 6. Our consistent orientation-driven arc costs for two different directions given the center node. INF indicates infinite values to prevent expansion in the lateral directions. $d$ is the Euclidean distance from a current node to a goal node.

## *B. Orientation-Driven Arc Cost*

We propose orientation-driven arc costs that depend on the orientation of the current node. This method is designed such that it provides a comfortable driving experience by restraining the steering actions of the vehicle. We also show how to make our arc costs adhere to the consistency condition that is useful for avoiding the reopening procedure in the A* algorithm.

We assign different values of arc costs to eight possible expansion nodes, depending on the orientation of the current node (Fig. 6). For movement straight forward where a child node has the same orientation as the current node, we can assign an arc cost equal to the diagonal grid width, say 1.4, which approximates $\sqrt{2}$ (not 1.0 for the movement straight forward), to the expansion between the current node and its expanded child node. This cost is chosen mainly because this cost should be valid when we have different orientations (see the right side of Fig. 6). For a 45° left or right turn, the arc cost can have a value greater than that for the straight forward movement, i.e., 1.4 to restrain the steering action. Given the constraint, we choose the value 2.0.

In order to show the consistency condition, let $c(n, n')$ and $c'(n, n')$ be arc costs defined by the Euclidean distance and our proposed method, respectively. By the aforementioned definition, $c(n, n') \leq c'(n, n')$. Therefore, our proposed arc costs satisfy the consistency condition because $h(n) \leq c(n, n') + h(n') \leq c'(n, n') + h(n')$ from (1). Since arc costs for expansion in the same orientation are smaller than those for expansion in different orientations, we can reduce unnecessary turns and construct a smooth trajectory. To movement in lateral directions from the current orientation we assign an infinite cost, thus blocking expansion to nodes that makes such lateral changes.

Unlike for forward movement, we restrain reverse movement when a vehicle is far from its goal. Specifically, the arc cost associated with reverse movement is multiplied by $d$, the Euclidean distance between the current node and the goal node. As a result, expansions for reverse movements are reduced because of higher arc costs. However, we cancel $d$ when its values becomes less than 1.5 times the length of the vehicle in order to equitably treat reverse and forward movements close to the goal. One can easily show that our orientation-driven arc costs are consistent.

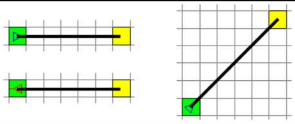
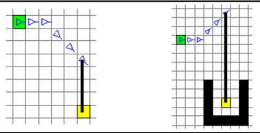
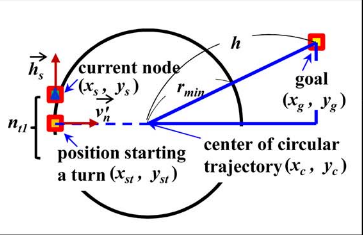
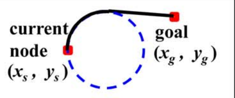
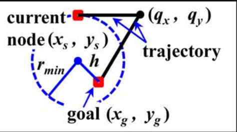
| Situation | Condition | Trajectory |
|---|---|---|
| 1)  | $\vec{h}_s = \vec{h}_g$<br>$\vec{h}_s$ : movement vector of current node<br>$\vec{h}_g$: vector from the current to the goal | — |
| 2)  | 1) $\cos^{-1}\left(\frac{\vec{h}_s \cdot \vec{h}_g}{\lvert\vec{h}_s\rvert\lvert\vec{h}_g\rvert}\right) = 45^\circ$ or $135^\circ$<br>2) Orientation is changeable | — |
| 3) The goal is located outside the turning radius<br>4) On the turning radius |  $\vec{h}_s$<br>current node<br>$(x_s, y_s)$<br>$h$<br>$r_{min}$<br>goal<br>$(x_g, y_g)$<br>$\vec{v}'_n$<br>$n_{tl}$<br>position starting a turn $(x_{st}, y_{st})$<br>center of circular trajectory $(x_c, y_c)$ |  |
| 5) Inside the turning radius | |  current node $(x_s, y_s)$<br>$(q_x, q_y)$<br>trajectory<br>$r_{min}$<br>$h$<br>goal $(x_g, y_g)$ |

Fig. 7. Conditions and types of trajectories for each situation in our obstacle-free kinematic heuristic.

## C. Obstacle-Free Kinematic Heuristic

Euclidean distance as a heuristic does not do well to represent a variety of situations, especially cases that depend on whether the goal is attainable within the turning radius of the vehicle, and ones where it is located in a direction lateral to that of the vehicle. In order to overcome these problems, we propose an *obstacle-free kinematic heuristic*, $H_{\text{free}}$, designed to compute the shortest path to goal without considering obstacles but taking into account the kinematics of the vehicle. The heuristic also supports efficient online computing.

To evaluate our obstacle-free kinematic heuristic, we compute the length of the shortest path based on the Dubins model, i.e., we calculate the optimal trajectory using a single line, and the combination of a circular arc and one or two lines [42]. We then use the length of the trajectory for the obstacle-free kinematic heuristic $H_{\text{free}}$.

Let us first define a movement vector as the orientation toward forward movement, or as its negative for the reverse movement. Different trajectory types can be constructed by considering the movement vector and whether the goal is within the turning radius of the vehicle. Based on these configurations, we generate five different cases and calculate trajectories in an obstacle-free environment (Fig. 7).

We can use a straight line for the trajectory in the following two cases: 1) when the movement vector of the current node $\overrightarrow{h_s}$ is identical to the vector from the current node to the goal node $\overrightarrow{h_g}$, or 2) when the angle between the two vectors $\overrightarrow{h_s}$ and $\overrightarrow{h_g}$ is $45^\circ$ or $135^\circ$, and we can expand nodes with diagonal edges.

Otherwise, we check the following three cases to calculate a trajectory based on a combination of a circular arc and a line (or two lines): 3) the goal is located beyond the minimum turning radius of the vehicle in its current position, 4) it is located on a trajectory with the minimum turning radius, or 5) it is inside the minimum turning radius. These five conditions and the corresponding computed paths are summarized in Fig. 7.

The latter three cases are determined by considering geometric relations, as shown in the condition at the bottom of Fig. 7. Given the minimum turning radius $r_{\min}$ and the distance $h$ between the turning center and the goal, these three cases are determined when $r_{\min} < h, r_{\min} = h$, and $r_{\min} > h$, respectively. The trajectory computed for the third and fourth cases consists of a circular arc followed by a line, while two lines are used for the fifth case.

To identify such conditions, we need to compute the center, $(x_c, y_c)$, of the circular trajectory (Fig. 7). When a node expands from its parent node, we have the following relationship between the position $(x_s, y_s)$ of the current node and the position $(x_{st}, y_{st})$ where it begins to turn

$$(x_{st},\ y_{st}) = (x_s,\ y_s) + (0.5n_{t1} - n_{t2})\overrightarrow{h_s}$$

where

$$n_{t2} = \begin{cases} n_t, & \text{if } n_{t1} > n_t \\ n_{t1}, & \text{otherwise.} \end{cases} \tag{2}$$

Here, $n_t$ is the number of nodes that continuously have the same orientation and direction (e.g., forward or reverse), $\overrightarrow{h_s}$ is the unit movement vector of the current node, and $n_{t1}$ is the number of straight edges of our trajectory module for the turning circle. These geometric quantities are shown at the bottom condition of Fig. 7. We can easily derive the above equation by examining the location of the current node in the trajectory module while respecting the definition of our trajectory module (Fig. 3), and the orientation of the current node as tangential to the circular arc.

We observe that the center $(x_c, y_c)$ of the circular trajectory is on a normal vector, $\overrightarrow{v}'_n$, heading inside the circle (and toward the goal) computed at $(x_{st}, y_{st})$. Based on this observation, the center $(x_c, y_c)$ of the circular trajectory is then obtained as follows:

$$(x_c,\ y_c) = r_{\min}\overrightarrow{v}'_n + (x_{st},\ y_{st})$$

where

$$\overrightarrow{v}_n = \pm\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\overrightarrow{h_s}$$
$$\overrightarrow{v}'_n = \underset{\overrightarrow{v}_n}{\arg\min}\left(\lVert r_{\min}\overrightarrow{v}_n + (x_{st},\ y_{st}) - (x_g,\ y_g)\rVert\right). \tag{3}$$

Here, $(x_g, y_g)$ is the position of the goal node.

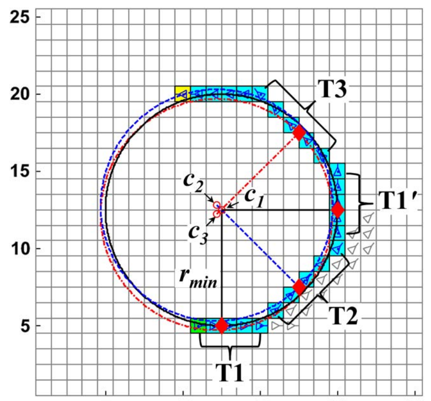


Fig. 8. This figure shows estimated centers of the circular trajectory. $(x_{st}, y_{st})$ begins the turn in each group (T1, T1′, T2, and T3) and are represented by ♦. The estimated center of each circular trajectory is placed on $c_1$ for T1 and T1′, $c_2$ for T2, and $c_3$ for T3. Lines shown in the figure represent $r_{\min}$ computed in each group. $r_{\min}$ is set to 7.5 m , $n_{t1}$ to 5, and the grid size to 1 m.

Fig. 8 shows the estimated center of the circular trajectory from the current expanded node. The centers are well estimated on the whole. In particular, the center of the trajectory is correctly computed on the horizontal or the vertical turning pattern, whereas it is less accurately computed for diagonal edges. This is mainly because the approximation of the circular trajectory based on our trajectory module is less accurate on diagonal edges. Note that the estimation of the center is used for the heuristic and not the calculation of the final path.

The fifth case represents a maneuver such as a one-point turn at $(q_x, q_y)$ [shown in Fig. 5(b)] for the situation where the goal is located inside the turning radius. We can simplify it by using two lines (the bottom trajectory in Fig. 7). This simplified approach satisfies the admissibility criteria for the heuristic, since the length of the lines is shorter than the length of a path generated in a grid space. $(q_x, q_y)$ is determined at a position where the current node can change its orientation.

The cost estimated by our obstacle-free kinematic heuristic is larger than or equal to that estimated by Euclidean distance and is also admissible, which will be discussed in Section IV-E; our obstacle-free kinematic heuristic makes nodes expanded fewer. Let two nodes exist. We assume that the Euclidean distance of the first node is larger than that of the second while the path costs of them are same, whereas the obstacle-free kinematic heuristic of the first is smaller than that of the second. In this situation, the second node can be selected as a parent if we use the Euclidean distance as a heuristic. However, the first node can be selected as a parent using the obstacle-free kinematic heuristic. It is desirable to select the node having smaller obstacle-free kinematic heuristic as the parent because the obstacle-free kinematic heuristic reflects the kinematics of the vehicle. As a result, it is more informative than the heuristic that uses Euclidean distance [40]. Fig. 9 shows the expanded nodes to find a path to goal using Euclidean distance as well as our obstacle-free kinematic heuristic. As expected, our heuristic identifies the path with fewer expanded nodes.

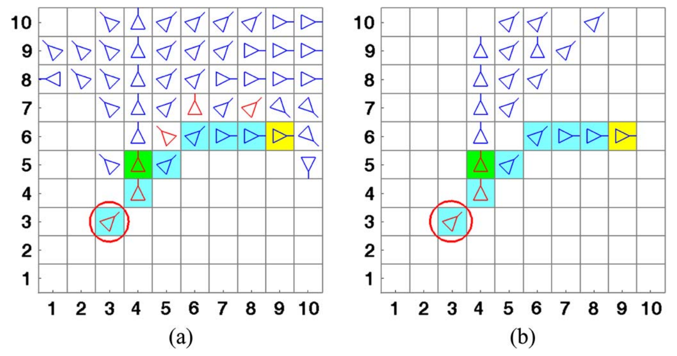


Fig. 9. Our obstacle-free kinematic heuristic (b) is more informative than one that uses Euclidean distance (a) $(n_{t1} = 2)$. In this example, a one-point turn is preferred to forward movement because space for forward movement is insufficient. The child nodes in red circles are included when calculating the final path, as shown in Fig. 5(b). (a) Euclidean. (b) $H_{\text{free}}$.

## *D. Obstacle-Aware Kinematic Heuristic*

Although the obstacle-free kinematic heuristic takes into account the kinematics of the car-like vehicle, it does not reflect obstacles in the environment and thus can fail to find a path to goal [Fig. 10(a)]. In order to overcome this problem, we propose an *obstacle-aware kinematic heuristic*, $H_{\text{obs}}$, obtained by computing the obstacle distance used in the geometric approach [43].

$H_{\text{obs}}$ is simply computed by modulating the obstacle-free kinematic heuristic $H_{\text{free}}$ with the inverse of the trajectory length (or obstacle distance), $\min_d$, from the current node to a position at which the vehicle collides with obstacles. We use the inverse of the trajectory length because the vehicle must avoid obstacles as it approaches them. To calculate the position of the vehicle at collision, we use the trajectories computed to calculate the obstacle-free kinematic heuristic (Fig. 7). Fig. 11 shows an example of a trajectory and $\min_d$ in the test environment.

Given the minimum distance $\min_d$ between the colliding points of potential obstacles, the obstacle-aware kinematic heuristic $H_{\text{obs}}$ is defined as follows:

$$H_{\text{obs}} = \begin{cases} \frac{H_{\text{free}}}{\min_d}, & \text{if } \min_d > 1 \\ H_{\text{free}}, & \text{if } \min_d \leq 1 \\ 0, & \text{no collisions.} \end{cases} \tag{4}$$

The first equation of (4) can overestimate the distance to goal on the grid map when $\min_d$ is too small, i.e., less than 1. We therefore limit its estimation with $H_{\text{free}}$ when $\min_d$ is less than one.

Fig. 10(a) and (b) shows expanded nodes for finding the path to goal. Compared to $H_{\text{free}}, H_{\text{obs}}$ finds the path more effectively, especially when it needs to be computed in the presence of obstacles.

## *E. Admissibility of Our Heuristics*

The proposed heuristics naturally satisfy the first and second conditions (finite successors and positive costs) of the three conditions of admissibility. We now discuss the conservativeness of the proposed heuristics. The obstacle-free kinematic heuristic is based on the optimal path to goal while

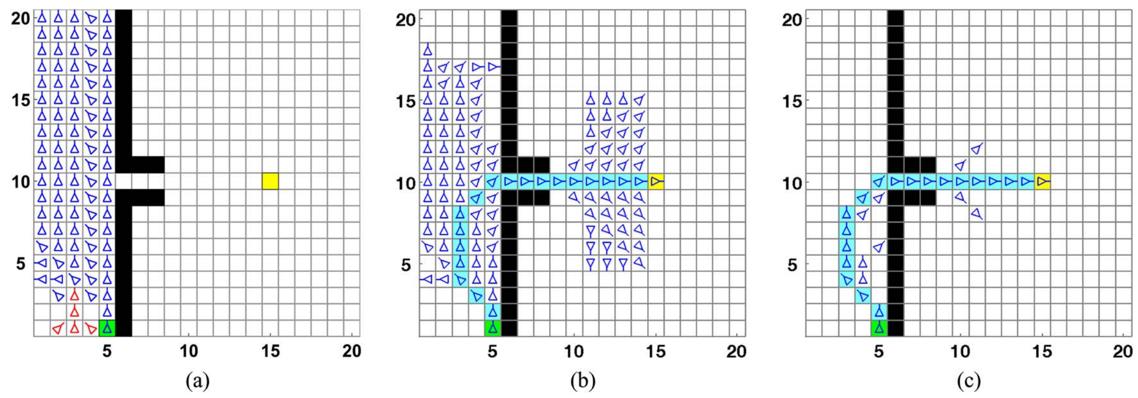


Fig. 10. (a) and (b) Show node expansion results combined only with the obstacle-free or obstacle-aware kinematic heuristic. (c) Shows the result combined with both. By using both heuristics, we can effectively find a path to goal. To test our method, $n_{t1}$ is set to 2, and the weight factor of the combined heuristic $k_h$ is set to 1.5. (a) $H_{\text{free}}$. (b) $H_{\text{obs}}$. (c) $H_{\text{free}}$ & $H_{\text{obs}}$.

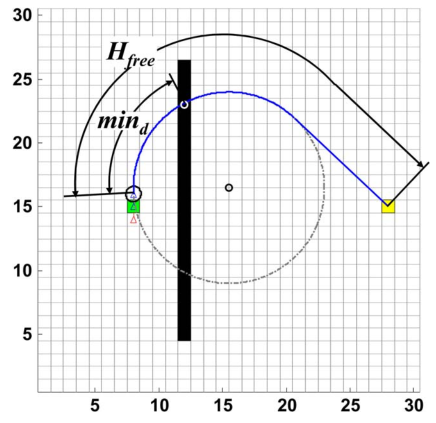


Fig. 11. A circular trajectory used to calculate the colliding point (shown in white circle) used in $H_{\text{obs}}$.

incorporating the kinematics of the car-like vehicle. It is admissible because it guarantees an estimated distance smaller than an actual distance to goal in the grid space. The obstacle-aware kinematic heuristic is also admissible because it only reduces the distance of the obstacle-free kinematic heuristic. Intuitively, if a distance estimated by any heuristic is larger than an actual distance, the evaluation cost $(f = g + h)$ at goal is higher than the cost of optimal path $(g^*(n_g))$ because the estimated heuristic at goal $(h(n_g))$ is higher than an actual distance $(h^*(n_g))$, i.e., zero. As a result, if any heuristic is not admissible, the found path is not optimal (see [40] for a formal proof).

The search procedures can be more efficient if they are combined. In general, a simple method for satisfying the admissibility of a heuristic is to compute a weighted sum of multiple admissible heuristics. When the sum of weights is less than or equal to 1, the combined heuristic is also admissible [44].

In the path finding problem, however, combining heuristics with a sum of weights of less than 1 has been known to be inefficient because the output of the combined heuristic is significantly smaller than the actual distance. To increase the search efficiency, we combine the two heuristics to form a new combined heuristic, $H^c_{\text{comb}}$, with a weight factor as follows:

$$H^c_{\text{comb}} = k_h(H_{\text{free}} + H_{\text{obs}}) \quad (5)$$

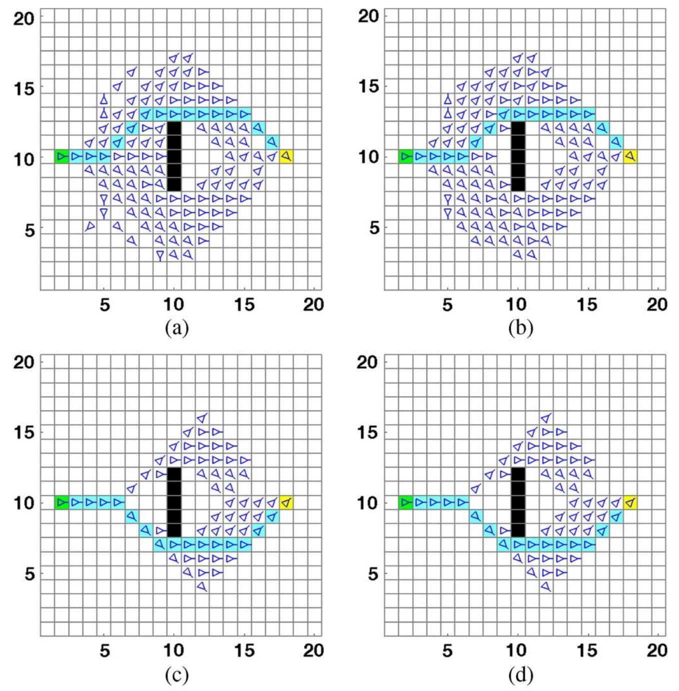


Fig. 12. Search patterns with different values of $k_h$. The values in parentheses indicate the number of expanded nodes. $n_{t1}$ is set to 2. (a) $k_h = 0.5$ (121). (b) $k_h = 1.0$ (120). (c) $k_h = 1.5$ (63). (d) $k_h = 2.0$ (63).

where $k_h$ is an amplification factor for the sum of the two heuristics. Note that we use superscript $c$ to indicate that we compute a path tracking the current orientation. We will also subsequently introduce a heuristic value $H^r_{\text{comb}}$, computed using an orientation opposite the current orientation in order to handle narrow passages.

When $H_{\text{obs}}$ is 0, i.e., there are no obstacles along the computed trajectory, we set $k_h$ to 1 to prevent the heuristic from overestimating the distance to goal. Otherwise, we aggressively set $k_h$ to be higher than 1 because it is more important in our problem to find paths of reasonable quality in an efficient manner than to find the optimal path. In practice, a range of 1.0 to 2.0, especially 1.5, for $k_h$ works well, and strikes a good balance between efficient search and high-quality paths.

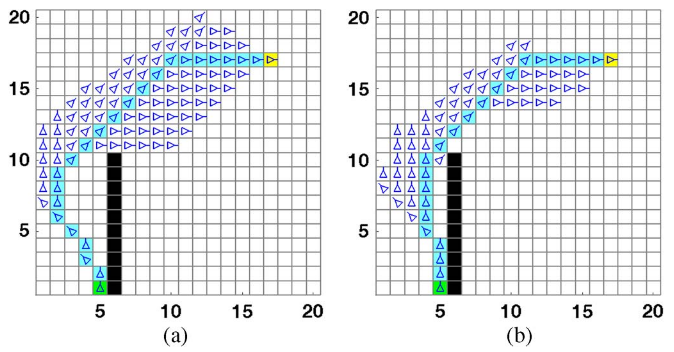


Fig. 13. Search patterns with different values of $n_{t1}$. $k_h$ is set to 1.5. (a) $n_{t1} = 2$. (b) $n_{t1} = 4$.

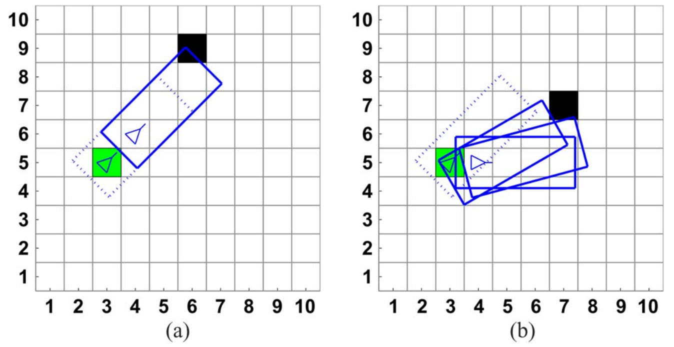


Fig. 14. Procedure for checking interference against obstacles when the vehicle moves straightforward or makes a turn. (a) Straightforward. (b) Turn.

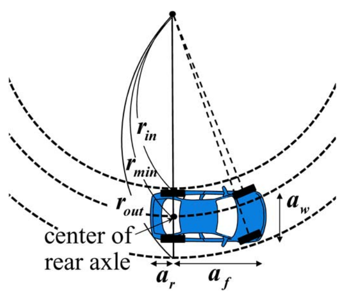


Fig. 15. The space required to make a turn. Our method computes such space while expanding nodes and evaluating heuristics.

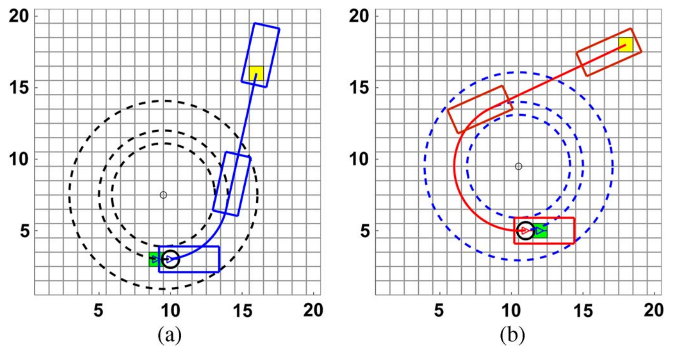


Fig. 16. Collision checking by considering the shape and turning space of the car-like vehicle. The solid line represents the trajectory for the vehicle while dashed lines represent the space required for a vehicle to turn. (a) Forward. (b) Reverse.

Fig. 12 shows search patterns with varying $k_h$ in a simple environment with obstacles. Moreover, Fig. 13 shows that when $k_h$ is higher than 1, our combined heuristic tends to be overestimated to a greater extent as we have a smaller value of the minimum turning radius. Nonetheless, it is more effective in environments containing complex obstacles, as shown in Fig. 10(c). In Section V-A, we show that our proposed method is more efficient than prior methods [19], [20].

### F. Shape-Aware Collision Checking

We have thus far discussed techniques considering the kinematics of car-like vehicles. In this subsection, we present a shape-aware collision checking technique that takes into account the shape of the vehicle. This method is based on our previous work that employs a graphical method [45]. As in that study, we approximate the shape of a vehicle as an oriented rectangle (Fig. 14), since it tightly approximates the shape of many types of vehicles. Unlike in our previous work, however, this method checks collisions along trajectories followed by or estimated by our kinematics-aware node expansion and heuristics while considering the shape and turning space of vehicles in order to reflect the kinematics of car-like vehicles.

We consider two cases, moving in a straight line or making a turn, where the shape of the vehicle is taken in account while following trajectories (Fig. 7). Checking interference against obstacles along a line is easily checked on each node during node expansion and on trajectories estimated by our heuristics, as shown in Fig. 14(a), by translating the box of the vehicle in the direction of movement.

When a car-like vehicle makes a turn, i.e., follows a circular arc, our approach considers the width and length of the vehicle, assuming that each node is placed on the center of the rear axle of the vehicle (Fig. 15). We then generate inner and outer circular arcs that cover the turning vehicle [46]. The turning radii of each of these inner and outer circular arcs is set to $r_{in}$ and $r_{out}$, respectively, based on the center of the rear axle, as follows:

$$\begin{aligned} r_{in} &= r_{\min} - 0.5a_w \\ r_{out} &= \sqrt{(r_{\min} + 0.5a_w)^2 + a_f^2} \end{aligned} \qquad (6)$$

where $a_w$ and $a_f$ are the width and the front overhang of the vehicle, respectively. Once we construct these inner and outer circular arcs, we check for interference against obstacles based on them (Fig. 16) with a higher accuracy irrespective of the resolution of the grids [47].

### G. Path Smoothing

In our node expansion, we can smooth a path by computing the turning center of circular arcs and placing the nodes for the path on the circular arcs along their normal direction. Since we compute the turning center of the vehicle from the center of the nodes, these turning centers may be computed a little differently, even from neighboring nodes, while making a turn as shown in Fig. 8.

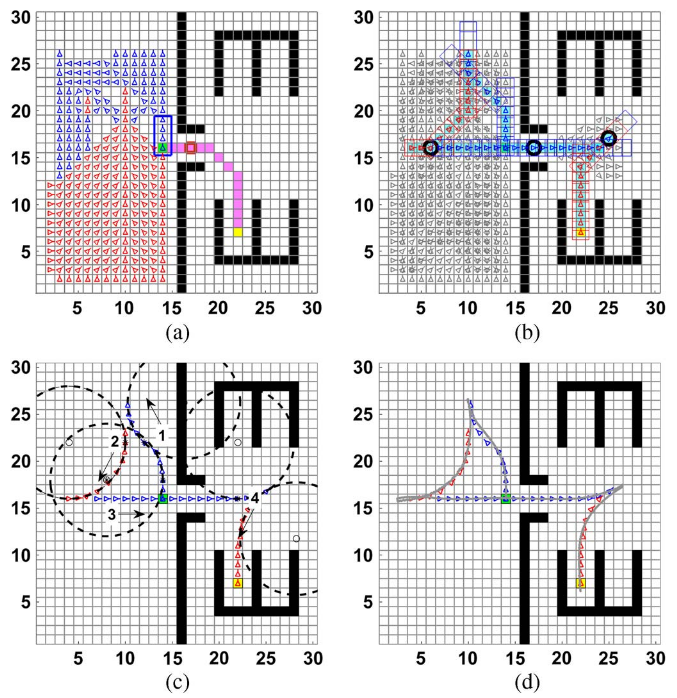


Fig. 17. We only use a state of the search space that encodes a position for the efficiency of search time. (a) shows an identified intermediate goal (shown in the red box) and (pink colored) path segments computed by our shape-aware A* for a narrow passage ($n_{t1}$ is set to 4). (b) shows a path found by the intermediate goal (coordinates (17, 16)) and nodes (coordinates (6, 16) and (25, 17)). (c) shows a smoothed path. Dashed circles represent the minimum turning radius of the vehicle. Numbers shown near the paths represent maneuvering sequence. (d) shows a path (shown in gray) tracked by our vehicle given a perfect obstacle map. Our path planner takes 91 ms on an Intel i7 computer to calculate the path.

To compute the turning center of the circular arcs, we access each node in the path and use the same turning center when making a consistent turn without any change of angle. We can assume that nodes prior to and following one that changes orientation make consistent turns. From the above process, in Fig. 8, we set the center of the circular arc to $c_1$ when nodes at T1 begin to turn. Fig. 17(c) shows the circular arcs of turning nodes and a smoothed path from an initial path [Fig. 17(b)] during our node expansion method.

## H. Intermediate Goals for Narrow Passages

Computing a collision-free path in environments with narrow passages poses a significant challenge for path planners. Similar problems arise when most prevalent A* algorithms are applied to narrow passages for a car-like vehicle. This is mainly because the vehicle needs to repeatedly make a forward/reverse movement to find a path to goal. Fig. 17(a) shows an example where it is difficult to find a path through a narrow passage owing to the kinematics of the vehicle.

In this kind of complex environment, we may need to pass the same region multiple times, i.e., have a node with the same state (we now use only a state of the search space that encodes a position for the efficiency of the search time) multiple times in a found path. Having a node with same state multiple times in a found path, unfortunately, is not allowed in the conventional A*. To allow this feature and effectively identify paths for narrow passages, we propose a recursive path-finding approach in multiple steps by introducing *intermediate goals* while using the reduced states of the search space. Finding intermediate paths using intermediate goals is a divide-and-conquer strategy for challenging environments. The divide-and-conquer algorithm works by breaking down a problem into sub-problems. The solutions to the sub-problems are combined to give a solution to the original problem. One may think of an alternative approach that uses higher-dimensional states of the search space, such as ones encoding orientations as well as positions to allow multiple nodes at the same position as long as they have different orientations. We attempted this alternative, but found that our current approach, which uses a simple state of the search space and recursive path-finding, is more efficient (Section V-A).

For our method, we first run our KSA* algorithm to find a collision-free path. When we cannot find such a path, we assume that the environment has a narrow passage. We perform two procedures to pass through the narrow passage: 1) we find an intermediate goal, $n_{mid}$, near the narrow passage, and 2) then find a path by passing from the intermediate goal to the final goal.

In order to find intermediate goals, we apply an A* algorithm, called *shape-aware A* algorithm*, that considers only the shape of the vehicle (Section IV-F). This shape-aware A* algorithm does not consider the kinematics of the car-like vehicle because we are interested mainly in checking whether the vehicle can pass through the narrow passage. A visual flow of the different components is shown in Fig. 2.

The path computed by the shape-aware A* algorithm is used only to compute intermediate goals for our KSA* algorithm, not as a path for the vehicle. We can then choose nodes along the identified path as intermediate goals for our KSA* algorithm. In particular, we choose the node from the identified path that is closest to expanded nodes of the prior operation of our KSA* algorithm. We choose the node (on the path found by the shape-aware A*) closest to expanded nodes among other possible candidates because there is a smaller probability that obstacles are encountered in regions close to nodes expanded by the KSA* algorithm.

Given the intermediate goal, we try to find a path to it by re-executing our KSA* algorithm. If we can find a path to the intermediate goal, we attempt to find a path from the intermediate goal to the final goal by running the KSA* algorithm again. Fig. 17(a) shows the identified intermediate goal in an environment with a narrow passage.

## I. Heuristic-Driven Search for Narrow Passage

We may not even find a path to an intermediate goal identified by our shape-aware A* method. In this case, we attempt to find a path to the intermediate goal by computing intermediate nodes. Unlike in the A* framework, we sort and search nodes based on heuristic values rather than evaluation costs similar to the best-first search (*heuristic-driven search*). This method is known to be very effective at expanding the most promising node to goal [40], i.e., it is suitable when we find a collision-free path for these difficult environments. Nonetheless, we could not prove

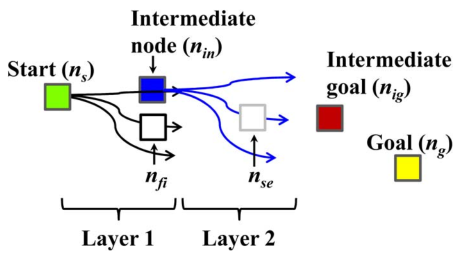


Fig. 18. Our heuristic-driven search method checks a path passing a new intermediate node $n_{se}$ by starting from a prior intermediate node $n_{in}$ or by checking another such node $n_{fi}$.

the completeness of our heuristic-driven search due to using the most promising nodes from the searching layers as shown in Fig. 18.

Thus far, we have computed $H^c_{\text{comb}}(n)$ for a node using the current orientation of node $n$. To increase a probability of finding a path to goal, we check to see whether we can compute a path to the node using the opposite orientation to the current one. The heuristic value of the reverse orientation is denoted by $H^r_{\text{comb}}(n)$.

In our heuristic-driven search, each node is associated with the minimum value $H^m_{\text{comb}}(n)$ between $H^c_{\text{comb}}(n)$ and $H^r_{\text{comb}}(n)$. Given two components $H_{\text{free}}(n)$ and $H_{\text{obs}}(n)$ of $H^m_{\text{comb}}(n)$, if $H_{\text{obs}}(n)$ is zero, it indicates a high probability that a collision will not occur on a path to goal.

To use this information, we sort values of $H^m_{\text{comb}}(n)$ for all expanded nodes in ascending order into two cases, depending on whether the value of $H_{\text{obs}}(n)$ is zero. We prioritize the $H^m_{\text{comb}}(n)$ for which the value of $H_{\text{obs}}(n)$ is zero over $H^m_{\text{comb}}(n)$ for which the value of $H_{\text{obs}}(n)$ is non-zero. We call $H^m_{\text{comb}}(n)$ sorted in this manner $H_{\text{sort}}(n)$. Note that this approach is different from that of A*, since we search nodes based only on heuristic values, not on evaluation costs. As a result, our heuristic-driven search cannot guarantee path optimality. Nonetheless, the environment that we are handling in this context is complex, and we thus focus on computing a collision-free path even though it is not optimal.

In this approach we choose a node with the minimum value of $H_{\text{sort}}(n)$ and treat it as an intermediate node, $n_{in}$. Note that we already have a path from the start node to $n_{in}$, since the node is chosen from expanded nodes from an earlier invocation of our KSA* algorithm. We then re-run the KSA* algorithm to find a path from the intermediate node $n_{in}$ to the intermediate goal $n_{ig}$.

It is possible that we are still unable to find a path between the intermediate node and the intermediate goal. Nonetheless, we have more information by now. First, we have expanded node information used to choose $n_{in}$; second, we have information about expanded nodes acquired to compute a path from $n_{in}$. We call nodes with the minimum value for each $H_{\text{sort}}(n)$ from the first and second pieces of information $n_{fi}$ and $n_{se}$, respectively. Of the nodes $n_{fi}$ and $n_{se}$, we choose the one with the minimum evaluation cost as the next intermediate node.

Intuitively, if we pick $n_{fi}$, we change $n_{fi}$ to $n_{in}$ and search another path from the start node (Fig. 18). If we otherwise pick $n_{se}$, we change $n_{se}$ to $n_{in}$ and continue to find a path starting

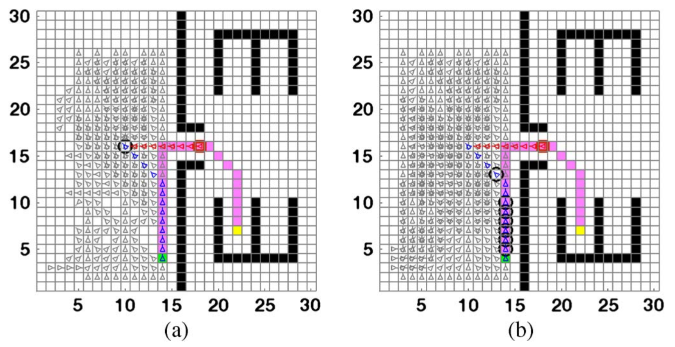


Fig. 19. Results of methods that compute intermediate nodes (shown in black circles) using our proposed heuristic values (a) and evaluation cost (b) to the intermediate goal (shown in the red box). The use of our heuristic values makes it possible to find a path to an intermediate goal with fewer expanded nodes or intermediate nodes than the method that uses evaluation cost.

from the prior intermediate node. We continue this process until we no longer have nodes in $H_{\text{sort}}(n)$. In other words, our recursive method terminates when no candidate nodes exist. As a result, our heuristic-driven search can efficiently reduce the number of intermediate nodes by using promising nodes instead of all nodes expanded from each intermediate node every time.

In order to show the benefits of our heuristic-driven search algorithm, we compared the expansion results of a method that uses our proposed heuristic values with one that considers evaluations costs (Fig. 19) in an environment with a narrow passage. As shown in the figure, our method finds a path to an intermediate goal with fewer expanded nodes or intermediate nodes. When evaluation cost is used as a heuristic, the method finds intermediate nodes close to the start node according to the common expansion pattern for any A* algorithm. Furthermore, there are cases where the common A* algorithm fails to find paths, whereas our recursive method succeeds in finding a path (Fig. 25). These results prove the efficiency and robustness of our method.

## V. Results

Thus far, we have shown the effectiveness of different components of our proposed algorithm through examples. In this section, we prove the efficiency of our two proposed online heuristics and the recursive path-planning method, followed by experimental results obtained by using an autonomous vehicle. The simulations and experiments are conducted using MATLAB, and our autonomous vehicle is equipped with controllers and a path planner, which are integrated using LabVIEW on an i7 computer with a 3.4 GHz CPU and 3 GB DRAM.

### A. Simulation Results

We tested our method against a complex scenario involving moving from a small region toward a goal in another tight region (Fig. 20). Our proposed method can find paths for such complex scenarios because of our recursive approach. Exiting a constricted space to reach the goal by passing a narrow passage, as shown in Fig. 20, is a difficult case because it requires for a node to repeatedly be in the same state in the found path, due

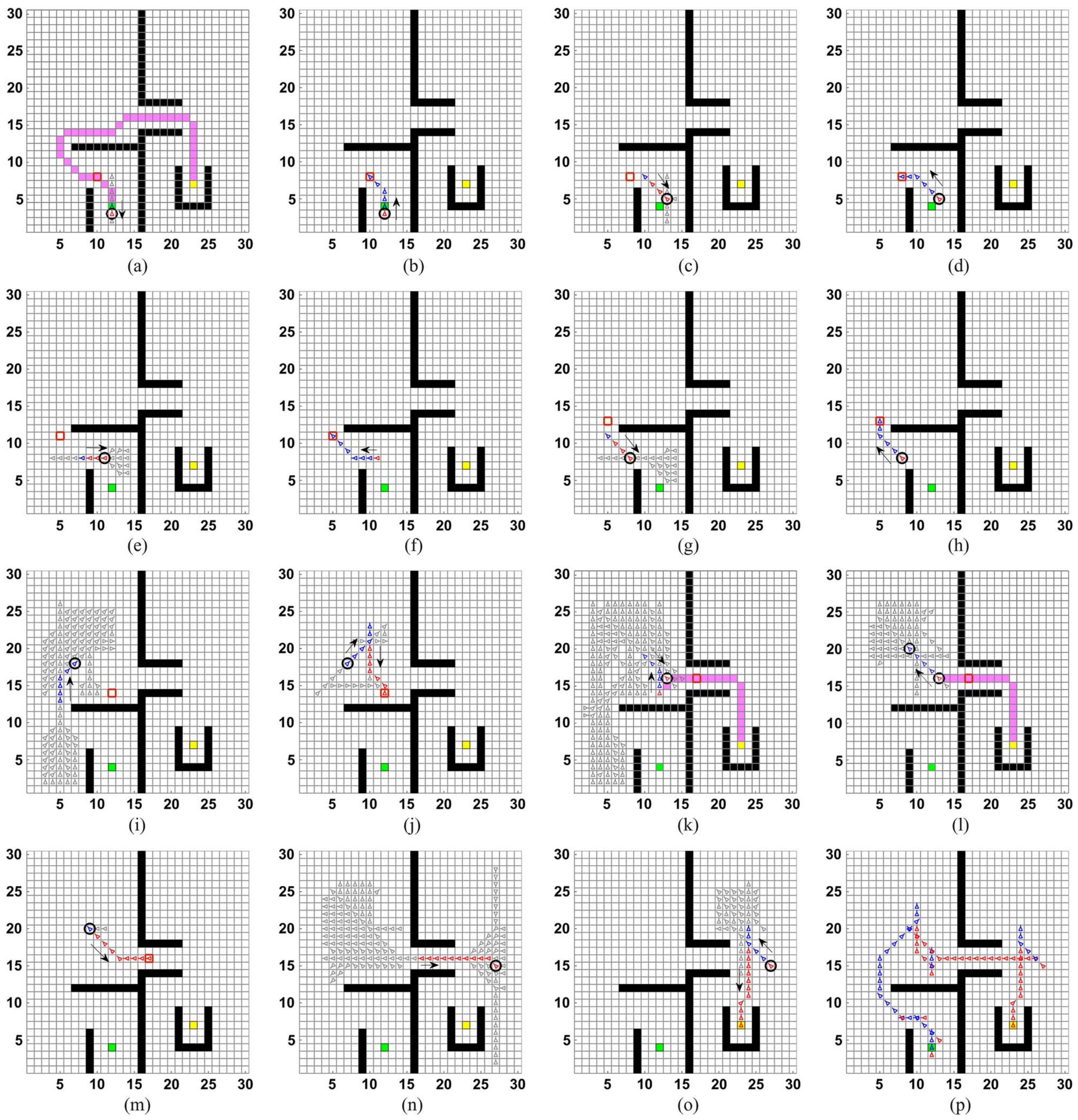


Fig. 20. (a) to (o) Show how our recursive method computes a path given a complex environment. (p) Shows the final path. Nodes in purple, in the red box, and in the black circle represent a path found by our shape-aware A*, intermediate goals, and intermediate nodes, respectively. Arrows indicate the direction of movement.

to repetitive forward and reverse movements of the vehicle in limited space.

In order to show the benefits of our proposed heuristics, we tested two versions, both of which use only our kinematics-aware node expansion. However, the first version uses previously proposed heuristics [19], [20] whereas the second one uses our heuristics. We performed these comparisons in two environments without and with obstacles (Fig. 21).

Table II shows the number of node expansions and the computation time required to evaluate them. Our method performs significantly better than the method that uses previously proposed heuristics across two benchmarks. This is primarily because our method is very efficient in computing our heuristics. On the other hand, the previously proposed heuristic functions, against which ours were compared, rely on expensive dynamic programming techniques for computing a heuristic for environments with obstacles. Our method also entails a smaller number of node expansions for environment with obstacles than the prior method. However, for a simple environment without obstacles [Fig. 21(a)], the prior method has a smaller number of expansions than ours. This is because the prior method uses dynamic programming that directly considers node expansions

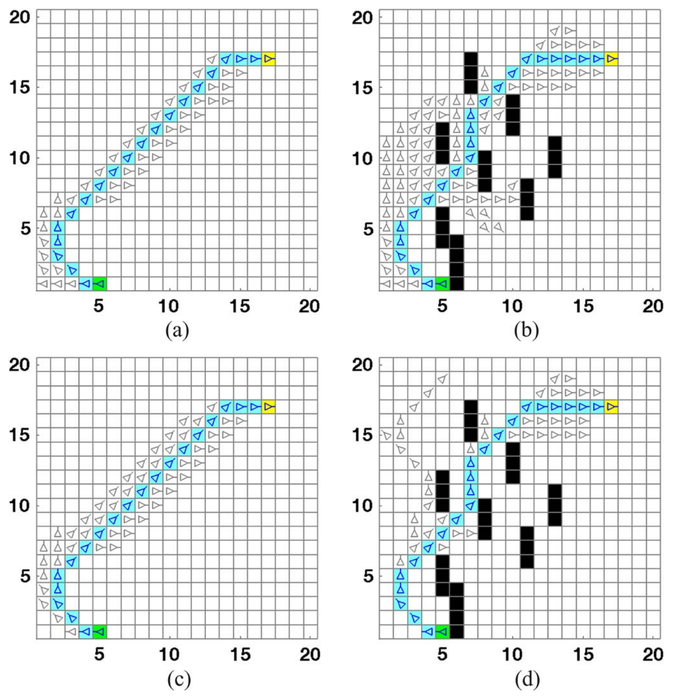


Fig. 21. (a) and (b) show expanded nodes and paths obtained by applying the maximum value of two heuristics proposed in [19], [20], and (c) and (d) are obtained by applying our two heuristics. $n_{t1}$ is set to 2.

TABLE II
THE NUMBER OF NODE EXPANSIONS AND AVERAGE COMPUTATION TIMES FOR 10 TRIALS USING MATLAB IN ORDER TO EVALUATE OUR HEURISTIC

| | Figs. 21(a) & 21(c) | | Figs. 21(b) & 21(d) | |
|---|---|---|---|---|
| | Prior [19], [20] | Ours | Prior [19], [20] | Ours |
| Computation time (s) | 58.87 | 1.31 | 75.44 | 1.23 |
| Num. of node expansions | 63 | 70 | 105 | 64 |

TABLE III
AVERAGE COMPUTATION TIME FOR 10 TRIALS USING MATLAB IN ORDER TO EVALUATE OUR RECURSIVE PATH-PLANNING METHOD. P AND O REPRESENT A POSITION AND AN ORIENTATION, RESPECTIVELY

| | State of P, w/ recursive method | State of P & O, w/o recursive method |
|---|---|---|
| Computation time (s) | 12.58 | 42.04 |

but does not consider the kinematics of the vehicle. As a result, in an empty environment, such as our test environment, dynamic programming can generate a path with a small number of node expansions. Nonetheless, in practice, there can exist many obstacles in a path, and the computation time for dynamic programming in such cases can be very expensive, as shown in this simulation.

*Detailed state representations:* One can represent states of the search space for nodes by using position and orientation, not only by using position as we did for our method. In this case, one may think that we can naturally allow multiple nodes at the same position as long as they have different orientations. As a result, we can avoid using the proposed recursive path planning. However, we found that this alternative is much slower than our proposed method. In order to show the efficiency of our recursive method in a simple representation of the state of the search space encoding only positional information, we compared the computation time of our proposed method with an alternative method encoding both position and orientation and not executing the recursive path-planning algorithm in the environment shown in Fig. 17. The runtimes of these methods are summarized in Table III. Fig. 22 shows a path with expanded nodes when we differentiate states of the search space by using position and orientation without executing the recursive planning algorithm. Our method can efficiently find feasible paths while expanding fewer nodes than the alternative method, which encodes both position and orientation without executing recursive planning.

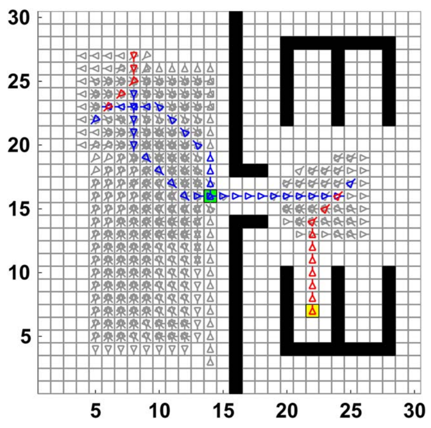


Fig. 22. This shows a found path and expanded nodes when we used states encoding positions and orientations without our recursive method.

## B. Experimental Results

We performed experiments on a ground vehicle (Fig. 1). The ground vehicle was equipped with a gasoline engine of 1591 cm$^3$, a four-speed automatic transmission, a front-wheel drive, and an anti-lock braking system (ABS). Its wheelbase was 2.55 m long (total length: 4.105 m) and the average tire angle for the two wheels was 26°. Based on the Ackerman steering condition, the minimum turning radius of the vehicle is computed as follows [46]:

$$r_{\min} = \frac{\text{wheelbase}}{\tan(\text{average tire angle})}. \tag{7}$$

From (7), $r_{\min}$ is approximately 5.2 m.

For all experimental results, we set the size of the grid cell in each dimension to 1 m. In this grid resolution, we set $n_{t1}$ to 4 and the turning radius $r_{\min}$ to 6 m. In order to achieve a more realistic turning radius for the test vehicle, we can use a higher-resolution grid map, i.e., for $r_{\min} = 5.2$ m, the ratio between the minimum turning radius in a 1-m grid and that of the real vehicle can be selected as the grid resolution (0.87 m per grid). However, we finally set the resolution to 1 m because it worked well in our experiments. Further, Dolgov *et al.* obtained good results with a 1-m resolution in the DARPA Urban Challenge [20].

For autonomous driving, the vehicle is installed with laser scanners, cameras, an inertial navigation system (INS), PCs, and actuators (Fig. 1). The in-vehicle actuators considered here included a motor-driven power steering wheel (MDPS), a gas pedal, brakes, transmission controlled by a controller-area network (CAN), a cruise control box, a rotary direct current (DC) motor, and a linear DC motor. Note that the laser scanners installed in the vehicle could detect obstacles in front of it and in a small lateral range ($\pm 135^\circ$ from the longitudinal axis) of the vehicle. We chose its field of view because it is sufficient to avoid obstacles placed in front of the vehicle while driving. In the case of reverse parking, we can sufficiently scan obstacles by moving forward and detect them using rear ultrasonic sensors while making a parking. Moreover, we acquired the position, steering angle, and speed of the vehicle using a differential global positioning system (DGPS) with a Pacific Crest Positioning Data Link Low-power base (PDL LPB) and sensors installed by the car manufacturer (Kia Motors Corp.).

We first tested how well our autonomous ground vehicle could compute and track a collision-free path. We placed the vehicle in an open space but used a perfect obstacle map of a virtually created complex environment with a narrow passage, as shown in Fig. 17(a). The path tracked by our vehicle given the perfect obstacle map is shown in Fig. 17(d). Our autonomous ground vehicle followed the computed path consisting of combinations of forward and reverse movements. The computation time taken by our path planner, installed in the autonomous ground vehicle, for a virtual but complex environment was 91 ms using an i7 computer.

*Environment with narrow passage:* We also tested our method in a real environment with a narrow passage. Fig. 23 shows that our vehicle passed the narrow passage by making a tight turn. Note that such a tight turn was made by taking into account the shape and turning space of the vehicle. The driving in this environment required reversing because obstacles were placed inside the minimum turning radius of the vehicle. Fig. 24 shows still images of this scenario.

*Recursive Path Planning:* We have pointed out that by encoding more information regarding the state of the search space of each node, we can find paths without using our recursive path planning as mentioned above. To show the significant benefits of our recursive path planning, we tested our method in another constricted region (i.e., parking in a narrow region) (Fig. 25). In this environment our recursive path planner found feasible paths while only using a simple state of the search space, i.e., encoding a position for each node. On the other hand, we could not find a path even though we used states of the search space encoding position, orientation, and direction of the vehicle. In this alternative method, we found paths for three sequences from 1 to 3, since the states of the nodes in these sequences are different. However, the nodes in the sequence 4 have the same state as the nodes in the sequence 2 because the reverse movement in those sequences occurs two times. This result shows the efficiency and robustness of our method for the planning problem of car-like vehicles. Fig. 26 shows still images of this scenario.

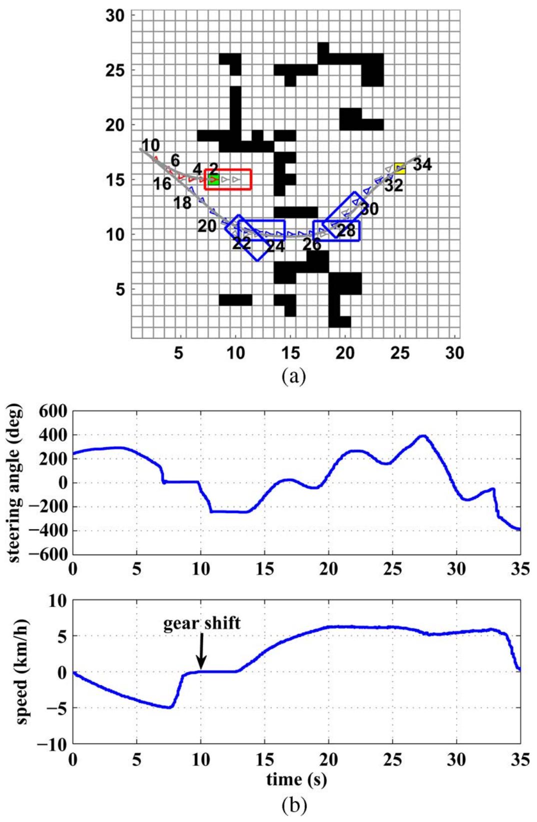


Fig. 23. (a) Our method generated a trajectory by making a tight turn and passing through a narrow passage. The number beside each path indicates the time (in seconds) when the vehicle passed a designated point on the path. (b) Steering angle and speed of the vehicle over time. The computation time was 4 ms.

## VI. Conclusion

In this paper, we proposed a holistic approach, the KSA* algorithm, to find a collision-free path to goal by taking into account the kinematics, shape, and turning space of a car-like vehicle. Our grid-based path planner used only a state of the search space encoding a position, without orientation and direction for the sake of efficiency. Specifically, we designed a kinematics-aware node expansion algorithm with orientation-driven arc costs for considering the kinematics of vehicles. We then proposed two online kinematic heuristics with obstacle-free and obstacle-aware approaches for efficient search within our KSA* algorithm. For complex environments with many obstacles and narrow passages, our method attempts to find a path by recursively identifying intermediate goals and nodes.

To verify the benefits of our method, we tested each component of our method using various simulations, and proved that our planner can support complex maneuvering, such as two-point turns in constricted space. Further, we tested our method in environments with narrow passages and confirmed that it satisfactorily handles such cases. Moreover, we compared our heuristics with state-of-the-art heuristics to exhibit the superior efficiency of our techniques. We also tested our path-planning method with an autonomous vehicle that captures obstacle maps in real time. Our results showed that the vehicle can follow paths generated by our method, which can compute paths even for environments with narrow passages.

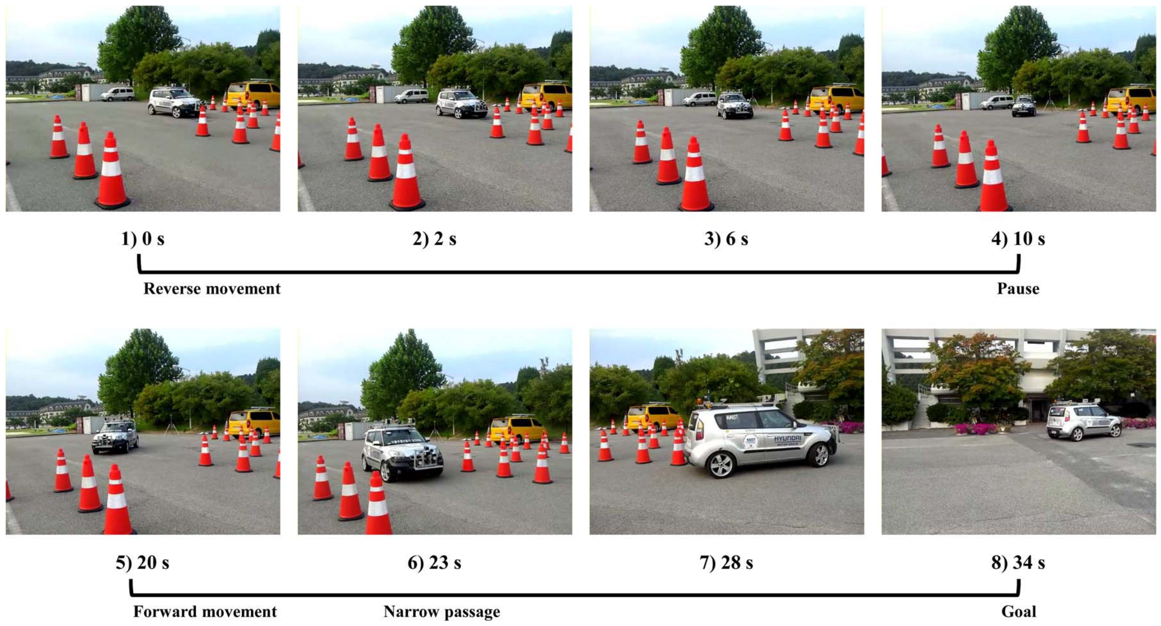


Fig. 24. An image sequence of passing through the narrow passage shown in Fig. 23. The vehicle made a reverse movement of up to 10 s, followed by a forward movement with a left turn.

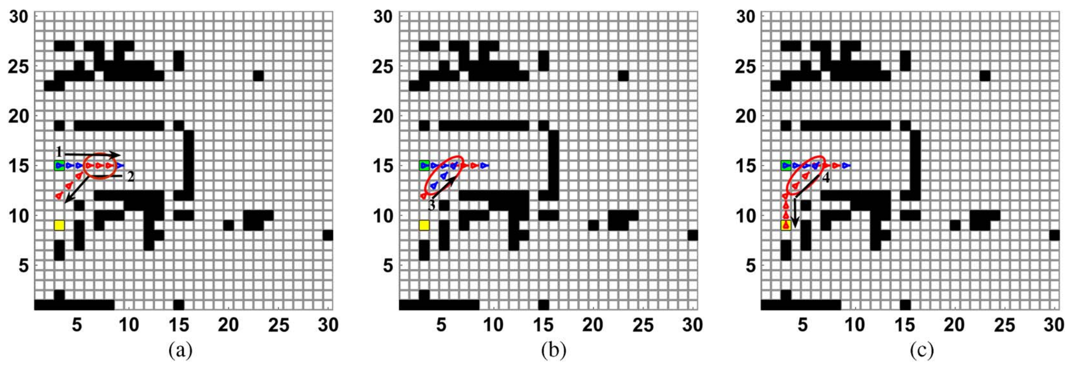


Fig. 25. These figures show that our recursive path planner with a state of search space encoding the position of each node can find feasible paths. It is difficult to find a feasible path, even though we used states of search space encoding position, orientation, and direction of the vehicle within the A$^*$ algorithm without running our recursive path planning. That is mainly because the sequence in 4 passes the nodes with the same state to ones of the sequence generated in 2. In other words, the reverse movement in that sequence occurred two times. The computation time of the path planner installed on our autonomous vehicle was 9 ms.

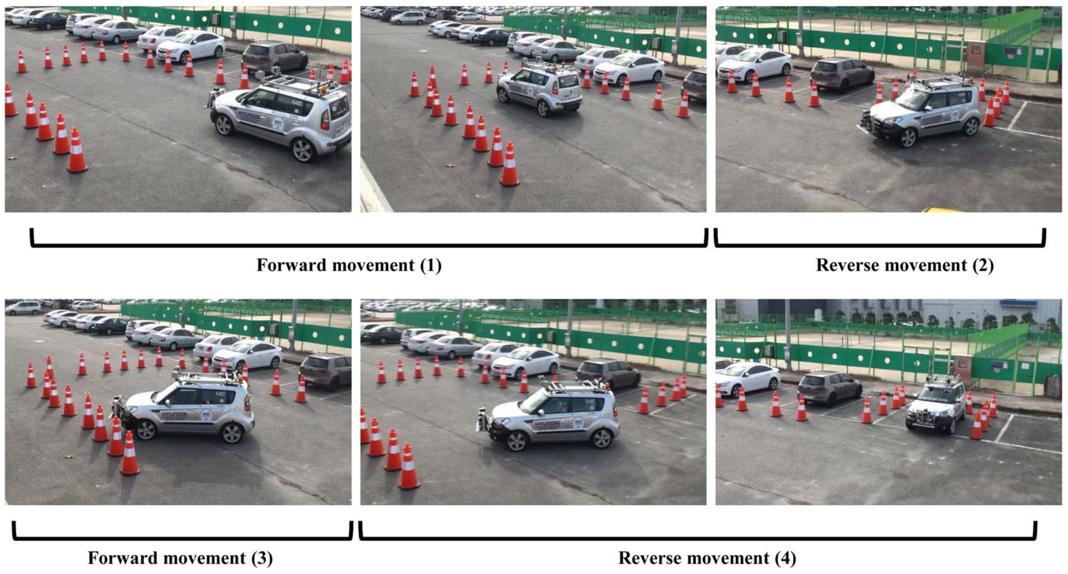


Fig. 26. An image sequence showing an advantage of our recursive path-planning method in a tight region, whose observed grid map is shown in Fig. 25.

Our research here open several interesting research avenues for future work. In this paper, we have not taken into account the speed of the vehicle. For situations such as a lane change during high-speed driving, the speed of the vehicle should be taken into account in computing the turning radius, which was assumed to be constant in this paper. Fortunately, our trajectory module can be dynamically re-computed according to the speed of the vehicle. Nonetheless, it remains for us to improve our method to operate robustly even in high-speed driving mode. In particular, we would like to consider the dynamics of the vehicle to support high-speed driving. We have shown here that our method is efficient and robust, but have not proved its completeness. We leave this pending for future work.

## REFERENCES

[1] U. Özgüner, C. Stiller, and K. Redmill, "Systems for safety and autonomous behavior in cars: The DARPA Grand Challenge experience," *Proc. IEEE*, vol. 95, no. 2, pp. 397–412, Feb. 2007.

[2] B. Siciliano, O. Khatib, and F. Groen, Eds., *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*. Berlin, Germany: Springer-Verlag, 2009.

[3] A. Fisher, *Inside Google's Guest to Popularize Self-Driving Cars*. New York, NY, USA: Popular Science, Sep. 2013.

[4] J. Wit, C. D. Crane, and D. Armstrong, "Autonomous ground vehicle path tracking," *J. Robot. Syst.*, vol. 21, no. 8, pp. 439–449, Aug. 2004.

[5] G. Oriolo, G. Ulivi, and M. Vendittelli, "Real-time map building and navigation for autonomous robots in unknown environments," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 28, no. 3, pp. 316–333, Jun. 1998.

[6] R. C. Luo and C. C. Lai, "Multisensor fusion-based concurrent environment mapping and moving object detection for intelligent service robotics," *IEEE Trans. Ind. Electron.*, vol. 61, no. 8, pp. 4043–4051, Nov. 2013.

[7] E.-H. Shin, "Accuracy improvement of low cost INS/GPS for land applications," M.S. thesis, Dept. Geomatics Eng., Univ. Calgary, Calgary, AB, Canada, 2001.

[8] C. R. Jung and C. R. Kelber, "A lane departure warning system using lateral offset with uncalibrated camera," in *Proc. IEEE Conf. Intell. Transp. Syst.*, Sep. 2005, pp. 348–353.

[9] S. S. Ge, X. Lai, and A. A. Mamun, "Boundary following and globally convergent path planning using instant goals," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 35, no. 2, pp. 240–254, Apr. 2005.

[10] H. Choset *et al.*, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. Cambridge, MA, USA: MIT Press, 2005.

[11] J. Gerdes and J. Hedrick, "Vehicle speed and spacing control via coordinated throttle and brake actuation," *Control Eng. Pract.*, vol. 5, no. 11, pp. 1607–1614, Nov. 1997.

[12] G. Antonelli, S. Chiaverini, and G. Fusco, "A fuzzy-logic-based approach for mobile robot path tracking," *IEEE Trans. Fuzzy Syst.*, vol. 15, no. 2, pp. 211–221, Apr. 2007.

[13] S. M. LaValle, M. S. Branicky, and S. R. Lindemann, "On the relationship between classical grid search and probabilistic roadmaps," *Int. J. Robot. Res.*, vol. 23, no. 7/8, pp. 673–692, Aug. 2004.

[14] P. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

[15] P. E. Hart, N. J. Nilsson, and B. Raphael, "Correction to: A formal basis for the heuristic determination of minimum cost paths," *SIGART Bull.*, vol. 37, pp. 28–29, Dec. 1972.

[16] D. Ferguson and A. Stentz, "Using interpolation to improve path planning: The field D$^*$ Algorithm," *J. Field Robot.*, vol. 23, no. 2, pp. 79–101, Feb. 2006.

[17] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta$^*$: Any-angle path planning on grids," *J. Artif. Intell. Res.*, vol. 39, pp. 533–579, 2010.

[18] J. Ziegler and M. Werling, "Navigating car-like robots in unstructured environments using an obstacle sensitive cost function," in *Proc. IEEE Intell. Veh. Symp.*, Jun. 2008, pp. 787–791.

[19] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *Int. J. Robot. Res.*, vol. 28, no. 8, pp. 933–945, Aug. 2009.

[20] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *Int. J. Robot. Res.*, vol. 29, no. 5, pp. 485–501, Apr. 2010.

[21] R. Geraerts and M. H. Overmars, "The corridor map method: A general framework for real-time high-quality path planning," *Comput. Animation Virtual Worlds*, vol. 18, no. 2, pp. 107–119, May 2007.

[22] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A$^*$," *Artif. Intell.*, vol. 155, no. 1/2, pp. 93–146, May 2004.

[23] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Trans. Robot.*, vol. 21, no. 3, pp. 354–363, Jun. 2005.

[24] B. Graf, J. M. H. Wandosell, and C. Schaeffer, "Flexible path planning for nonholonomic mobile robots," in *Proc. 4th Eur. Workshop Adv. Mobile Robots*, 2001, pp. 199–206.

[25] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[26] D. Hsu, J.-C. Latombe, and H. Kurniawati, "On the probabilistic foundations of probabilistic roadmap planning," *Int. J. Robot. Res.*, vol. 25, no. 7, pp. 627–643, Jul. 2006.

[27] S. M. LaValle and J. J. Kuffner, Jr., "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, May 2001.

[28] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011.

[29] J. L. Ny, E. Feron, and E. Frazzoli, "On the Dubins traveling salesman problem," *IEEE Trans. Autom. Control*, vol. 57, no. 1, pp. 265–270, Jan. 2012.

[30] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT$^*$," in *Proc. IEEE ICRA*, May 2011, pp. 1478–1483.

[31] R. A. Knepper and A. Kelly, "High performance state lattice planning using heuristic look-up tables," in *Proc. IEEE/RSJ IROS*, 2006, pp. 3375–3380.

[32] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *Amer. J. Math.*, vol. 79, no. 3, pp. 497–516, Jul. 1957.

[33] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners," in *Proc. IEEE ICRA*, Sep. 2003, vol. 3, pp. 4420–4426.

[34] L. Zhang, Y. J. Kim, and D. Manocha, "A hybrid approach for complete motion planning," in *Proc. IEEE/RSJ IROS*, Oct. 2007, pp. 7–14.

[35] S. Dalibard and J.-P. Laumond, "Linear dimensionality reduction in random motion planning," *Int. J. Robot. Res.*, vol. 30, no. 12, pp. 1461–1476, Oct. 2011.

[36] J. Lee, O. Kwon, L. Zhang, and S.-E. Yoon, "A selective retraction-based RRT planner for various environments," *IEEE Trans. Robot.*, vol. 30, no. 4, pp. 1002–1011, Aug. 2014.

[37] N. I. Katevas, S. G. Tzafestas, and C. G. Pnevmatikatos, "The approximate cell decomposition with local node refinement global path planning method: Path nodes refinement and curve parametric interpolation," *J. Intell. Robot. Syst.*, vol. 22, no. 3/4, pp. 289–314, Apr. 1998.

[38] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi, "A stable tracking control method for a non-holonomic mobile robot," in *Proc. IEEE/RSJ Int. Workshop Intell. Robots Syst.*, Nov. 1991, vol. 3, pp. 1236–1241.

[39] A. P. Aguiar and J. P. Hespanha, "Trajectory-tracking and path-following of underactuated autonomous vehicles with parametric modeling uncertainty," *IEEE Trans. Autom. Control*, vol. 52, no. 8, pp. 1362–1379, Aug. 2007.

[40] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, MA, USA: Addison-Wesley, 1984.

[41] N. J. Nilsson, *Artificial Intelligence: A New Synthesis*. Burlington, MA, USA: Morgan Kaufmann, 1998.

[42] F. Lamiraux and J.-P. Laumond, "Smooth motion planning for car-like vehicles," *IEEE Trans. Robot. Autom.*, vol. 17, no. 4, pp. 498–502, Aug. 2001.

[43] M. Vendittelli, J.-P. Laumond, and C. Nissoux, "Obstacle distance for car-like robots," *IEEE Trans. Robot. Autom.*, vol. 15, no. 4, pp. 678–691, Aug. 1999.

[44] L. Mandow and J. P. de la Cruz, "Multicriteria heuristic search," *Eur. J. Oper. Res.*, vol. 150, no. 2, pp. 253–280, Oct. 2003.

[45] S. Yoon and D. H. Shim, "SLPA$^*$: Shape-aware lifelong planning A$^*$ for differential wheeled vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 2, pp. 730–740, Apr. 2015.

[46] R. N. Jazar, *Vehicle Dynamics: Theory and Applications*. Berlin, Germany: Springer-Verlag, 2008.

[47] D. Kim, J.-P. Heo, J. Huh, J. Kim, and S.-E. Yoon, "HPCCD: Hybrid parallel continuous collision detection using CPUs and GPUs," *Comput. Graph. Forum (Pacific Graphics)*, vol. 28, no. 7, pp. 1791–1800, Oct. 2009.

**Sangyol Yoon** received the B.S. degree in mechanical engineering from Hongik University, Seoul, Korea, in 1999, the M.S. degree in mechatronics from Gwangju Institute of Science and Technology, Gwangju, Korea, in 2001, and the Ph.D. degree from Korea Advanced Institute of Science and Technology, Daejeon, Korea, in 2014.

From 2003 to 2007, he was with Samsung Electronics Company Ltd., Suwon, Korea, where he was engaged in the development of three-axis optical pickup actuators for optical disk drives. From 2007 to 2009, he was with Hyundai Mobis, Yongin, Korea, where he was involved in pre-crash and advanced external airbag systems development. Since 2014, he has been with LG Electronics Inc., Incheon, Korea. His research interests include vehicle motion planning and control, and active safety.

**Sung-Eui Yoon** (SM'13) received the B.S. and M.S. degrees in computer science from Seoul National University, Seoul, Korea, in 1999 and 2001, respectively, and the Ph.D. degree in computer science from the University of North Carolina at Chapel Hill, NC, USA, in 2005. He is currently an Associate Professor at Korea Advanced Institute of Science and Technology, Daejeon, Korea. He was a Postdoctoral Scholar at Lawrence Livermore National Laboratory. He co-wrote a monograph on real-time massive model rendering and gave numerous tutorials on proximity queries and large-scale rendering at various conferences, including ACM SIGGRAPH and IEEE Visualization. His main research interests include designing scalable graphics and geometric algorithms. Dr. Yoon is a member of the ACM and Eurographics. He was a recipient of a Distinguished Paper Award at Pacific Graphics, invitations to IEEE TVCG, an ACM Student Research Competition Award, and other domestic research-related awards.

**Unghui Lee** received the B.S. degree in aerospace engineering from Korea Aerospace University, Goyang, Korea, in 2009 and the M.S. degree in aerospace engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2011. He is currently working toward the Ph.D. degree in aerospace engineering with KAIST. His research interests include path planning and navigation for autonomous systems.

**David Hyunchul Shim** received the B.S. and M.S. degrees in mechanical design and production engineering from Seoul National University, Seoul, Korea, in 1991 and 1993, respectively, and the Ph.D. degree in mechanical engineering from the University of California, Berkeley, CA, USA, in 2000.

From 1993 to 1994, he was with Hyundai Motor Company, Seoul, as a Transmission Design Engineer. From 2001 to 2005, he was with Maxtor Corporation, Milpitas, CA, as a Staff Engineer. From 2005 to 2007, he was with the University of California as a Principal Engineer in charge of the Berkeley Aerobot Team. In 2007, he joined the Department of Aerospace Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. He is currently an Associate Professor at the Department of Aerospace Engineering, College of Engineering, KAIST, and the Director of the Center of Field Robotics at KAIST. His interests include control theory, unmanned aerial vehicles, self-driving cars, and field robotics.