International Journal of Automotive Technology, Vol. 16, No. 4, pp. 653−668 (2015)
DOI 10.1007/s12239−015−0067−5

pISSN 1229−9138/ eISSN 1976−3832

REAL-TIME PATH PLANNING OF AUTONOMOUS VEHICLES FOR UNSTRUCTURED ROAD NAVIGATION

K. CHU[1)], J. KIM[1)], K. JO[2)] and M. SUNWOO[3)*]

[1)]ADAS Control Development Team, Automotive Research and Development Division, Hyundai Motor Group, 150 Hyundaiyeonguso-ro, Hwaseong-si, Gyeonggi 455-706, Korea
[2)]Department of Automotive Engineering, Graduate School, Hanyang University, Seoul 133-791, Korea
[3)]Department of Automotive Engineering, Hanyang University, Seoul 133-791, Korea

(Received 29 October 2013; Revised 30 July 2014; Accepted 27 October 2014)

ABSTRACT−This paper presents a path planning algorithm for autonomous vehicles to generate feasible and smooth maneuvers in unstructured environments. Our approach uses a graph-based search algorithm and a discrete kinematic vehicle model to find a primitive path that is non-holonomic, collision-free, and safe. The algorithm then improves the quality and smoothness of the identified path by making local shortcuts. This shortcut for smoothing the path is determined by connecting the internal state of the primitive path to the initial state based on the Pythagorean Hodograph (PH) cubic curve. The formulation of the PH cubic curves in the smoothing path provides arc-length parameterization of the curve in a closed form. We present examples and experimental results for the real time path planning in unstructured road from an implementation on an autonomous vehicle.

KEY WORDS : Autonomous vehicle, Graph-based search, Path planning, Pythagorean hodograph

1. INTRODUCTION

Research on the safety of vehicles has been done to develop more effective warning systems to avoid collisions and accidents. Many warning and driver assist functions have been integrated into current vehicles. A recent focus of active safety is to increase autonomous driving capability (Horowitz and Varaiya, 2000; Shladover, 1992; Robinson *et al*., 2010; Bergenhem *et al*., 2010). These technologies will be gradually developed from semi-autonomous driving to full autonomous driving for driving safety and comfort. The progress of autonomous vehicle technologies requires the ability to handle realistic road scenarios including exceptional driving situations (Bishop, 2005; Montemerlo *et al*., 2008; Kuwata *et al*., 2009; Levinson *et al*., 2011; Urmson *et al*., 2008). These needs require path planning for autonomous vehicles in urban driving scenarios for both high speed navigation for structured roads and complex navigation for unstructured roads.

Many studies have been successfully conducted on autonomous navigation for structured road scenarios (Chu *et al*., 2012; Thrun *et al*., 2006; Werling *et al*., 2010; Bacha *et al*., 2008; Leonard *et al*., 2008). On the other hand, research on free-form navigation for unstructured roads has not matured yet. In this case, a motion planning algorithm dedicated for free space navigation is required to generate a global path to the desired pose. This free-form path planner is necessary when the vehicle handles anomalous driving situations in structured roads (blocked lane road, partially blocked intersection) and requires a sequence of complex maneuvers.

Path planning for unstructured road navigation differs significantly from navigation in a lane. While driving in a lane implicitly provides the preferred path, there are no driving lanes in unstructured roads, and thus motion planning is far less constrained. For operating in complex environments, an autonomous vehicle has to use its on-board sensors and to generate maps of the environment because predefined road information is unavailable. For this reason, the path planning for unstructured roads is a complex and computationally expensive problem.

Path planning algorithms for free-form navigation have mostly been studied for robotic applications. Recently, automotive-oriented path planning problems have been studied for managing complex driving scenarios. The main challenges in developing free-form path planning are generation of continuous trajectories and their optimization while considering the non-holonomic constraint of a vehicle. Sample-based planning algorithms such as the probabilistic roadmap (PRM) and Rapidly-exploring Random Trees (RRT) have been widely studied to generate paths for free-form navigation (Karaman and Frazzoli, 2011; LaValle and Kuffner Jr, 2001; Kuwata *et al*., 2009).

**Corresponding author.* e-mail: msunwoo@hanyang.ac.kr

Sample-based planning algorithms use randomly generated samples to connect adjacent samples. These algorithms can guarantee kinematic feasibility by adapting a system model, but they require efficient guiding heuristics for practical online implementation. A randomized sampling strategy sometimes induces unnatural paths that may contain unnecessary turns or unexpected changes. Direct formulation of path planning as non-linear optimization does not guarantee fast convergence due to the complex optimization landscape with multiple local minima. Much of the prior work on heuristic search algorithms for path planning have yielded fast algorithms for discrete spaces, but these algorithms have produced non-smooth paths and they do not satisfy the non-holonomic constraints of vehicle motion (Likhachev *et al*., 2005; Ferguson and Stentz, 2005; Hart *et al*., 1968).

To account for non-holonomic constraints, several variants of grid-based search algorithms have been studied (Urmson *et al*., 2008; Pivtoraiko *et al*., 2009; Pivtoraiko and Kelly, 2005; Montemerlo *et al*., 2008; Ferguson *et al*., 2008; Likhachev and Ferguson, 2009). These approaches consist of two phases. In the first phase, the path planner obtains a primitive path by using discretization of the state and a grid. If solely applying search algorithms in a discretized state space, the primitive path is a kinematically feasible path but is non-smooth. If the discretized state space has more dimensions to improve the quality of the solution, the search algorithm requires exponential growth of the memory usage and computational load due to the incremental nature of the algorithm. In the second phase, therefore, a path planner improves the quality of the primitive path performing numerical optimization (Urmson *et al*., 2008; Pivtoraiko *et al*., 2009; Pivtoraiko and Kelly, 2005; Montemerlo *et al*., 2008; Ferguson *et al*., 2008; Likhachev and Ferguson, 2009; Dolgov and Thrun, 2009). The main challenges for numerical optimization are to compute collision-free paths guaranteeing numerical convergence and efficiency in a complex environment. In particular, gradient-based optimization does not work well without an initial guess since the constraints from environmental obstacles impose many local minima and can be non-smooth (Pan *et al*., 2012).

In this paper, we present path planning that generates a feasible path and improves the quality of the feasible path by using a Pythagorean Hodograph (PH) cubic curve. Firstly, a graph-based heuristic search is employed to produce a primitive path by discretizing and reducing the vehicle states. This primitive path is kinematically feasible, collision-free, and safe, due to the combination of a kinematically feasible graph, a risk map, and collision checking on the obstacle map. Secondly, instead of applying the numerical optimization of the path, a shortcut technique is adopted to guarantee the numerical stability sacrificing the optimality of the solution for the online implementation. The PH cubic curves are used to make shortcut on the primitive path by connecting an internal state of the primitive path to the initial state, which makes it possible to enhance the path quality related to kinematically feasible, smooth, and collision-free characteristics. The PH cubic curve provides the piecewise G^1 continuity and arc-length re-parameterization of the curve in a closed form (Jakliè *et al*., 2010). These properties of PH cubic curve simplify the formulations of the maneuvering for obstacle avoidance with smooth motion in terms of implementation. The proposed path-planning algorithms were implemented successfully on the autonomous vehicle A1 shown in Figure 1 which won both the 2010 and 2012 Autonomous Vehicle Competition (AVC) organized by the Hyundai Motor Group in Korea (Chu *et al*., 2011; Jo *et al*., 2013).

Figure 1. Autonomous vehicle A1.

The remainder of this paper is organized as follows. In Section 2, our graph-based incremental search algorithm is presented. Section 3 describes the path smoothing based on the Pythagorean Hodograph cubic curve. We next describe the online re-planning in Section 4 The simulation results and experimental results by implementing the autonomous vehicle A1 are presented in Section 5. Finally, we summarize and conclude this paper in Section 6.

2. GRAPH-BASED INCREMENTAL PLANNING

2.1. Graph Based Search Algorithm

In traditional grid-based approaches, obstacles in the environment are mapped to a set of cells, and each cell represents the presence of an obstacle at that position in the environment as shown in gray in Figure 2 (a). Optimal search methods, such as the A* and D* algorithms are generally used to find the globally sub-optimal path connecting each cell from an initial position to a goal while avoiding obstacles in the environment, as shown in Figure 2 (a) (Likhachev *et al*., 2005; Ferguson and Stentz, 2005; Hart *et al*., 1968). The resolution of the grid cells causes a tradeoff between the optimality of a path and the practical efficiency of computation and usage of memory to find a path. In the traditional grid-based approach, it is difficult to handle the non-holonomic constraint of the vehicular system.

In our approach, the non-holonomic constraint of a vehicle is satisfied to generate a path by executing the

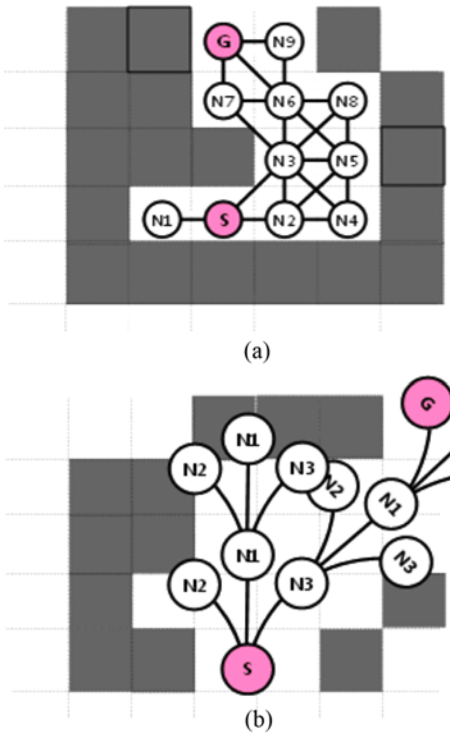

Figure 2. Graph representation of the search space. (a) Grid-map based representation of the graph (b) Kinematic motion based representation of the graph.

```
Algorithm 1: GraphSearch($G$, $\mathbf{q}_{init}$, $\mathbf{q}_{goal}$)
1:  $u_{init} \leftarrow (\mathbf{q}_{init}, G)$
2:  $u_{goal} \leftarrow (\mathbf{q}_{goal}, G)$
3:  $Q \leftarrow \{u_{init}\}$
4:  while $Q \neq \emptyset$ do
5:  $u \leftarrow \min_{u \in O} f(u)$
6:  $Q \leftarrow Q \setminus \{u\}$
7:    if $\|u - u_{goal}\|_2 \leq B_g$ then
8:  return ReconstructPath ($G$, $u$)
9:  endif
10: $M \leftarrow$ GenrateManeuver ($G$, $u$)
11: Add Maneuver ($G$, $u$, $M$)
12: end while
```

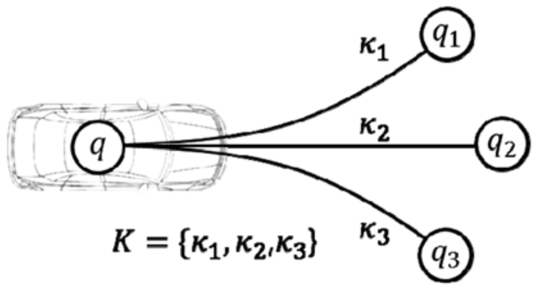

Figure 3. Example of the expanded state $\mathbf{q}_i$ generation.

kinematic vehicle model. For generation of the path, a graph data structure is generated independently from the obstacle grid map (Dolgov *et al.*, 2010; Pivtoraiko *et al.*, 2009; Likhachev and Ferguson, 2009; Pivtoraiko and Kelly, 2005). Each vertex u in the graph G is mapped by a vehicle state that consist of a position (x, y) and an orientation ψ, and the vehicle state is denoted as $\mathbf{q} = [x, y, \psi]^T$. The path generation starts with mapping the current state of the vehicle $\mathbf{q}_{init}$ and the goal state $\mathbf{q}_{goal}$ to the initial vertex u_{init} and the goal state u_{goal} respectively (Algorithm1, line 1-2). In next step, the initial vertex is added to the open-list Q, containing all the vertices sorted by $f(\cdot)$ (Algorithm1, line 3). At each step of the iteration, the next vertex at the top of Q is popped from Q and used for expansion of the graph (Algorithm 1, line 5). For expansion of the graph structure, new vehicle states are generated by several steering actions at Generate Maneuver (Algorithm1, line 10) and this composes the discrete search space as shown in Figure 2 (b).

The steering action is derived from the kinematic vehicle model which is given by

$$\frac{dx}{dt} = v\cos\psi,\ \frac{dy}{dt} = v\sin\psi,\ \frac{d\psi}{dt} = v\kappa, \tag{1}$$

where v is the vehicle speed, and κ is the curvature. The curvature of the motion is limited by the steering angle constraint.

```
Algorithm 2: GenenrateManeuver ($G$, $u$)
1: $M = \theta$
2: foreach $\kappa_i \in K$
3: q $\leftarrow (G, u)$
4: $\mathbf{q}_i = \mathbf{q} + \Delta\mathbf{q}_i$
5: $M = M \cup \{\mathbf{q}_i\}$
6: end
7: return $M$
```

```
Algorithm 3: AddManeuver ($G$, $u$, $M$)
1.  foreach q $\in M$ do
2:     if CollisionFree(q) then
3:   $u_{new} \leftarrow (\mathbf{q}, G)$
4:   $e \leftarrow (u, u_{new}, G)$
5:   $Q \leftarrow Q \cup \{u_{new}\}$
6:   UpdateCost(e, $u_{ne}$, $G$)
7:   else if
8:   Update Cost Collision (u, $G$)
9:      end if
10:  end
```

This constraint for the curvature is given by

$$|\kappa| \le \kappa_{\max}. \tag{2}$$

Suppose K is the predefined subset of the discrete values of the curvature and $\kappa_i \in K$. By assuming a constant vehicle speed, the difference equation of the steering action for each step can be represented with respect to the arc-length of a curve:

$$\begin{bmatrix} x_{i,s+\Delta s} \\ y_{i,s+\Delta s} \\ \psi_{i,s+\Delta s} \end{bmatrix} = \begin{bmatrix} x_s \\ y_s \\ \psi_s \end{bmatrix} + \begin{bmatrix} \cos(\psi)\ \Delta s \\ \sin(\psi)\ \Delta s \\ \kappa_i\ \Delta s \end{bmatrix} \Rightarrow \mathbf{q}_i = \mathbf{q} + \Delta\mathbf{q}_i, \tag{3}$$

where $[x_{i,s+\Delta s},\ y_{i,s+\Delta s},\ \psi_{i,s+\Delta s}]^T = \mathbf{q}_i$ is the expanded state with respect to κ_i, and $[x_s, y_s, \psi_s]^T = \mathbf{q}$ is the state associated with the parent vertex u (Algorithm 2, line 3-4). Figure 3 depicts an example of expanded state generation using the kinematic vehicle model and the finite curvature set K. For each curvature κ_i, a new state $\mathbf{q}_i$ is added to the maneuver set M. The generation process of the maneuver is summarized in Algorithm 2. The generated maneuver that is the set of new states is used for growing the graph data structure $G = (V, E)$, where V is the set of vertices and E is the set of edges, through the Add Maneuver (Algorithm 1, line 11). In the Add Maneuver, new vertices and edges are assigned to the graph G, if the new state is obstacle-free (Algorithm 3, line 2-4). At the same time, the new vertices are inserted to an open-list Q (Algorithm 3, line 5). When the new state has a collision with an obstacle, the state is discarded and the cost of the neighbor vertex is updated for penalization (Algorithm 3, line 8). For evaluation of each vertex, the cost is updated by using several grid-based maps that are described in the next section (Algorithm 3, line 6).

This incremental graph-search scheme provides a primitive path that is a sequence of vehicle states $P_{pri} = \{\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_N\}$, when it meets the condition of path finding. This primitive path is kinematically feasible because it guarantees a minimum turning radius of vehicle motion. Furthermore, it indicates a collision-free path, as shown in Figure 4.

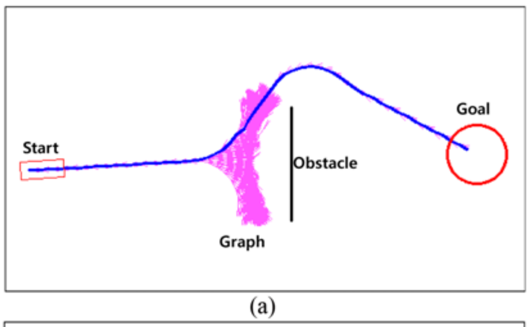

Figure 4. Primitive path.

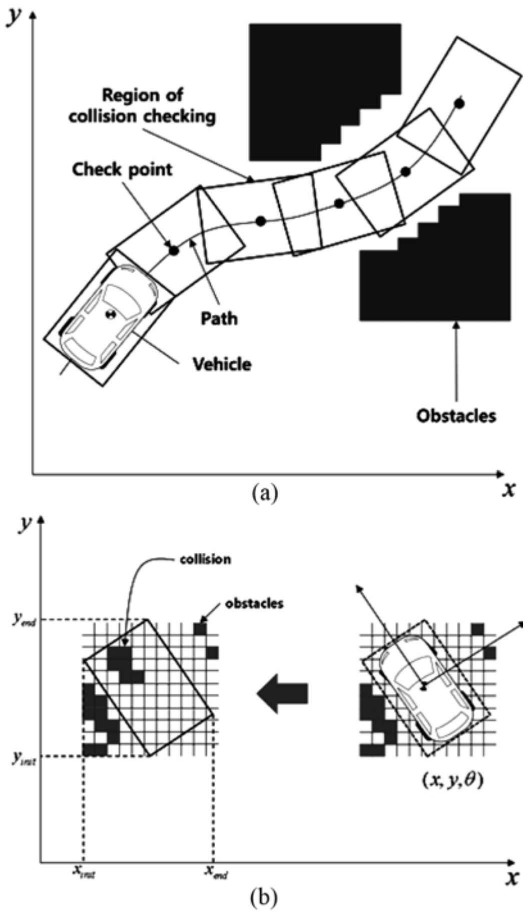

Figure 5. Collision check: (a) Collision checking for the graph; (b) Region of interest for collision checking with respect to a vertex of the graph.

2.2. Obstacle Grid-map and Collision Check

An obstacle map specifies the adjacent obstacles in the local coordinates with a 20 cm resolution. The physical constraints including static obstacles and road boundaries are represented in the obstacle map. Each grid-cell contains the obstacle information in binary data. This obstacle map can be overlaid on the graph G to represent a drivable area with respect to which we want to find a drivable path as shown in Figure 5. In the path planning problem of the autonomous vehicle, collision checking is frequently considered as one of the major bottlenecks for real-time computation. In particular, the orientation and the shape of a vehicle should be considered. In many robotic applications, the shape of the robot is regarded as a disk and the orientation is neglected for simplicity. However, the vehicle orientation and shape are very important for checking the collision for the automotive application, because the drivability of the road relies on the current vehicle orientation and the shape of the vehicle.

Figure 5 (b) illustrates the region of collision check for a given path and the reason why the shape and orientation of the vehicle should be considered for collision checking. Approximation of the vehicle shape to a disk simplifies computation for the checking of collision, since a disk is invariant to rotation and the orientation of the vehicle is not considered. However, since the ratio of the total length and the total width of the vehicle is approximately 5 to 2, and the approximation of a vehicle using a disk is not suitable to check the collision of the path for passing the narrow road, as shown in Figure 5 (a). Therefore, the vehicle shape is regarded as a rectangle to consider the vehicle orientation and vehicle shape. In order to reduce the computational load for exhaustive search in the discrete obstacle grid map, the region of interest for the collision check is determined for each check point as shown in Figure 5 (b).

Each check point provides the position and orientation of the vehicle $[x,\ y,\ \psi]^T$ and the region of interest for collision checking can be determined by considering the vehicle shape. The region of interest is given by

$$x_{init} \le x \le x_{end},\ y_{init} \le y \le y_{end}. \tag{4}$$

The limit of the region of interest is determined as follows:

$$\begin{aligned} x_{init} &= \min(x_{FL}, x_{FR}, x_{RL}, x_{RR}) \\ x_{end} &= \max(x_{FL}, x_{FR}, x_{RL}, x_{RR}) \\ y_{init} &= \min(y_{FL}, y_{FR}, y_{RL}, y_{RR}) \\ y_{end} &= \max(y_{FL}, y_{FR}, y_{RL}, y_{RR}), \end{aligned} \tag{5}$$

where (x_{FL}, y_{FL}) is the position of the front left corner of the vehicle, (x_{FR}, y_{FR}) is the position of the front right corner of the vehicle, (x_{RL}, y_{RL}) is the position of the rear left corner of the vehicle, and (x_{RR}, y_{RR}) is the position of the rear right corner of the vehicle. In order to check for a collision of the path, all of the cells that are occupied by an obstacle within the region of interest should be investigated. If the occupied cell exists in the region of interest, the given path has a collision.

The problem of collision checking is converted to determine the existence of an occupied cell in the polygon. The way of finding whether a point exists inside a polygon is to test how many times a ray starting from the point intersects the edges of the polygon, as shown in Figure 6 (Stein, 1997). If the point is in the outside of the polygon, the number of crossing the boundary of the polygon is an even number including zero. On the other hand, the number of intersecting the edge of the polygon is an odd number.

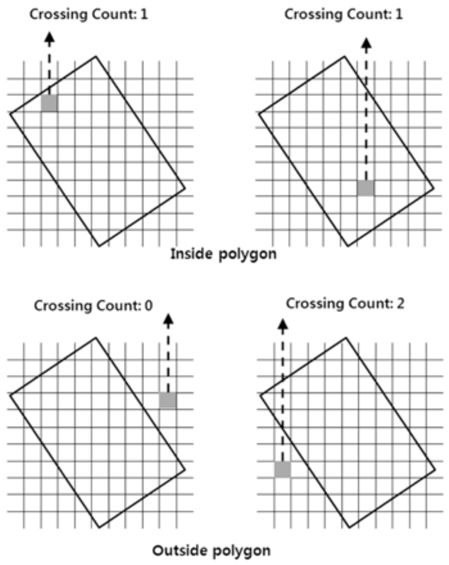

Figure 6. Ray casting for collision check.

2.3. Cost Function

This section describes how the vertices in the graph are sorted when the graph G is expanding toward the goal (Algorithm1, line5). The graph expansion is guided by the total cost $f(\cdot)$ corresponding to each vertex that is the weighted sum of accumulated distance from the initial state, the cost-to-go, risk cost, and collision cost are as follows:

$$f(u) = w_{coll}\mu_{coll} + w_{risk}\mu_{risk} + w_{dist}\mu_{dist} + w_{heur}\mu_{heur} \tag{6}$$

where μ_{coll} is the collision cost, μ_{risk} is the risk cost, μ_{dist} is the accumulated distance from the initial state to the state according to the edges of the graph, and μ_{heur} is the heuristic cost. After the propagation of new states, the resulting states are evaluated with obstacles and a cost that is represented to several grid-based maps.

2.3.1. Heuristic cost and accumulated distance

In many path planning applications, shortest path criterion is generally adopted because the collision probability can be lower within the shorter path. The most simple way to measure the distance to the goal from every point is the two-norm distance in the 2D space. Although the two-norm distance in the 2D space does not consider the non-holonomic constraint of the vehicle motion, it provides an approximated distance to the goal state and efficient guidance to the search algorithm at a small difference of orientations between the goal state and every state. However, accurate representation of the distance is required for accurate path planning problems such as parking. The path planning for parking is beyond the scope of this paper. For this reason, the Euclidean distance is only considered as a cost-to-go heuristic cost as follows (Likhachev *et al.*, 2005; Ferguson and Stentz, 2005; Hart *et al.*, 1968):

$$\mu_{huer}(u) = \sqrt{(x(u) - x_{goal})^2 + (y(u) - y_{goal})^2}. \tag{7}$$

The accumulated distance is simply computed by the accumulation of the distance between the vertices from the initial vertex. The accumulated distance is determined by

$$\mu_{dist}(u) = \mu_{dist}(u_{parent}) + \Delta s, \tag{8}$$

where u_{parent} is the parent vertex of the current vertex u (Likhachev *et al.*, 2005; Ferguson and Stentz, 2005; Hart *et al.*, 1968).

2.3.2. Risk cost

Although a collision check for each vertex is required to prevent collision, it is insufficient to remove the vertex that has a collision for generating a safe path. If the shortest path criterion is solely applied to the graph-based path planning, the path can be generated adjacent to obstacles as shown in Figure 7 (b).

The proximity of obstacles for autonomous driving may increase the probability for collision caused by control errors and occluded obstacles. It is difficult to select a safer vertex from the graph by solely checking the collision as shown in Figure 7 (a) because the collision check only provides the binary information. Therefore, a risk-cost map is used for penalizing the proximity to obstacles in this approach. The risk cost around an obstacle occupying the grid-cell is determined by using the distance transform. The distance transform of the binary grid-map specifies the distance from each cell to the nearest non-zero cell (P. F. Felzenszwalb and Huttenlocher, 2004). Rather than an obstacle grid-map that specifies the presence or absence of an obstacle at each cell, it can be useful to have a map that

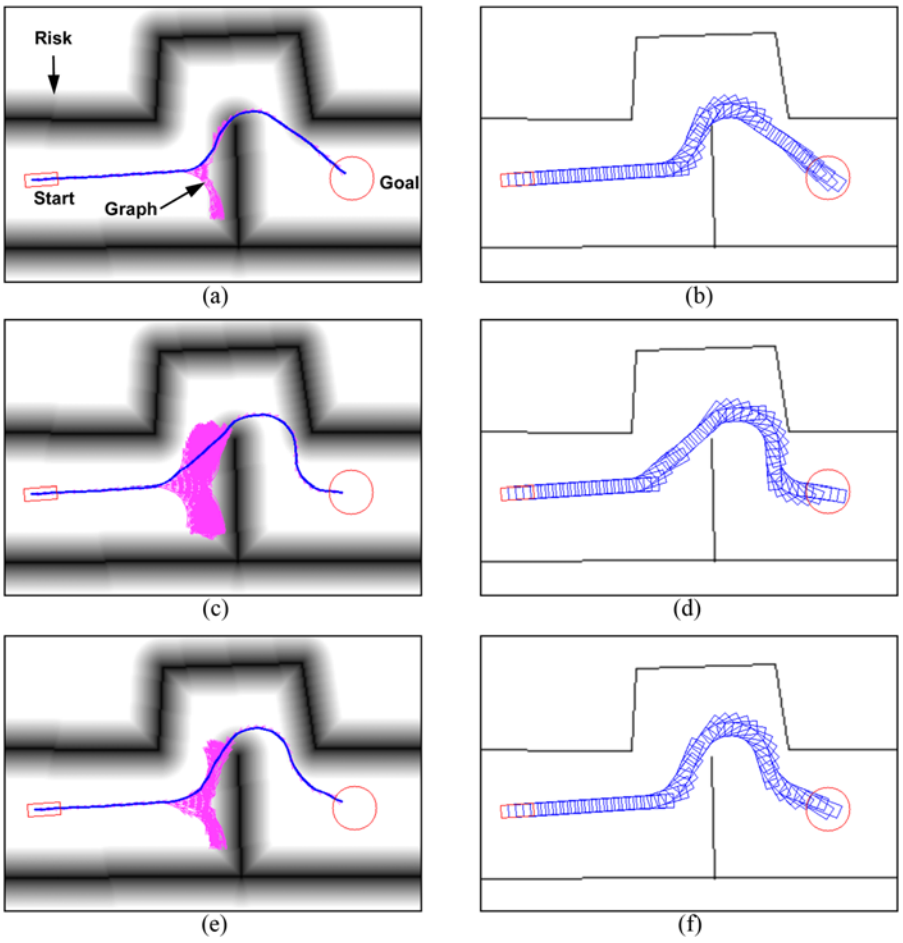

Figure 7. Effect of cost: (a) Graph expansion guided by the accumulated distance and the heuristic cost; (b) Primitive path guided by the accumulated distance and the heuristic cost; (c) Graph expansion guided by a combination of the accumulated distance, the heuristic cost, and the risk cost; (d) Primitive path guided by a combination of the accumulated distance, the heuristic cost, and the risk cost; (e) Graph expansion guided by a combination of the accumulated distance, the heuristic cost, the risk cost, and the collision cost; (f) Primitive path guide by a combination of the accumulated distance, the heuristic cost, the risk cost, and the collision cost.

specifies the risk for the proximity of an obstacle at each cell.

The risk-map is closely related to the distance transform of a set of point on an obstacle grid-map O, which associates to the distance of the nearest obstacle location p_O in O. Given the point $\mathbf{p}=(x, y)$ in 2D space, we compute the distance to the nearest obstacle, $D_O(x, y)=D_O(\mathbf{p})=\min_{P_O\in O} d(\mathbf{p},\mathbf{p}_O)$, where $\mathbf{p}_O$ is the position of an obstacle on the obstacle map and $d(\mathbf{p}, \mathbf{p}_O)$ is the measure of the distance between $\mathbf{p}$ and $\mathbf{p}_O$. The risk associated with a given point is defined as follows:

$$\mu_{risk}(x,y)=\begin{cases}\dfrac{D_{\max}-\gamma D_O(x,y)}{D_{\max}} & \text{if } \dfrac{D_O(x,y)}{\alpha}<D_{\max},\\ 0 & \text{elsewhere,}\end{cases} \tag{9}$$

where $\gamma > 0$, $D_{\max} > 0$ are the control parameters that adjust the decreasing rate of risk and the maximum effective range of the risk.

2.3.3. Collision cost

If the collision of a state is detected during the graph expansion, the collision detected states are not assigned to the vertex of the graph and the probability of a collision is propagated to other vertices that have the same parent vertex of the collision detected state (Algorithm 3, line 8). This propagation is computed as follows:

$$\mu_{coll}=\frac{N_c}{N_\kappa}, \tag{10}$$

where N_C is the collision count within the maneuver from the parent vertex, and N_κ is the expanded number of the maneuvers from the parent vertex. Penalizing the neighborhood of the collision state can reduce the function calls for the computationally expensive collision detection. Therefore, an incremental path planner can search the free space without wasting the vertices during the expansion of graph.

2.3.4. Effect of cost

Figure 7 illustrates the simulation results for the presentation of the effects of the costs. In Figure 7 (a), the heuristic cost and the accumulated distance are considered for the graph expansion. The number of the expanded vertices of the graph is not large, but the resulting path shows a very sharp turn around the obstacles and is very close to an obstacle, as shown in Figure 7 (b). These types of paths may cause hug wall behavior of vehicle motion that is undesirable. In Figure 7 (c) and Figure 7 (d), the effect of the risk cost is presented by adding to the heuristic cost and the accumulated distance.The risk is presented as the gray color of the map as shown in Figure 7 (c). The dark area presents the risky area and the light area is the safe zone. Weighting for the risk cost can prevent driving at the proximity of an obstacle, whereas it may cause a large number of graph expansions. The increase in the graph expansion requires a long computation time to find a path. Thus, the collision cost is additionally combined for the same risk cost to weaken the increase of the graph expansion penalizing the vertex that is adjacent to the collision as shown in Figure 7 (e). The generated path with total cost including the collision cost is shown in Figure 7 (f).

3. SMOOTHING BASED ON PYTHAGOREAN HODOGRAPH

The path reconstructed from the graph is drivable, satisfying the kinematic constraint, but it consists only of a full turn and a straight motion. For this reason, the path solely constructed from the graph is not easy to track by a vehicle control system, and can cause an uncomfortable motion of the vehicle.

To improve the quality of the generated path, post-processing for the solution of the incremental search algorithm is performed after reconstructing the path. In this section, we introduce our smoothing approach based on the Pythagorean Hodograph (PH) cubic curve. Our approach uses PH cubic curves to connect from the current states to the state of the primitive path, which makes it possible to smooth the primitive path.

3.1. Pythagorean Hodograph

PH curves form a special subclass of the planar polynomial parametric curves that have expression of the arc-length using polynomial function of parameter (Jaklič *et al.*, 2010; Alves Neto *et al.*, 2010; Farouki and Sakkalis, 1990). The representations of the curves are exclusively parametric formulations based on the cubic polynomial function in the Bernstein-Bezier form as

$$\mathbf{r}(t)=\{x(t), y(t)\}=\Sigma_{k=0}^{n}\mathbf{b}_k B_k^n(t),\ \text{for } t\in[0,1] \tag{11}$$

where $B_k^n(t)$ Bernstein polynomial given by

$$B_k^n(t)=\binom{n}{k}(1-t)^{n-k}t^k,\ k=0,\ldots,n \tag{12}$$

where $\boldsymbol{b}_k$ are the control points for $k=0,\ldots,3$ (Farouki and Sakkalis, 1990). This parametric form can be easily rendered by uniformly increasing the parameter t and evaluating the polynomial in (9). However, when a curve is rendered by evaluation at a uniform sequence of parameter t, the resulting geometric point is not uniformly spaced along the curve, since its parametric flow is necessarily uneven if it is not merely a straight line. Therefore a functional relation between the arc-length along the curve and the parameter value t has to be determined. Pythagorean Hodographs are the first derivative of parametric polynomials satisfying the Pythagorean condition. The hodograph of a planar curve that is the first derivative of the curve (9) is given by

$$r'(t)=\{x'(t),y'(t)\}. \tag{13}$$

The hodograph $\mathbf{r}'(t)$ of a polynomial curve $\mathbf{r}(t)$ is said to be Pythagorean if its components are members of a

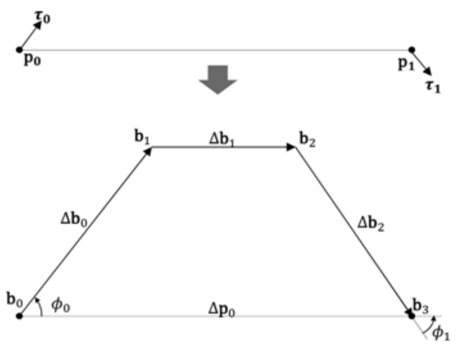

Figure 8. Control polygon for the PH cubic curve with the two given points p_0, p_1 and their tangent directions.

Pythagorean polynomial triple $(x'(t), y'(t), \sigma(t))$ as follows (Farouki and Sakkalis, 1990):

$$x'(t)^2+y'(t)^2=\sigma(t)^2 \tag{14}$$

The control point of a cubic Bezier curve can be represented as

$$\mathbf{b}_0=\mathbf{p}_0,\ \mathbf{b}_1=\mathbf{p}_0+\lambda_0\tau_0,\mathbf{b}_2=\mathbf{p}_1+\lambda_1\tau_0,\ \mathbf{b}_3=\mathbf{p}_1 \tag{15}$$

where $\mathbf{p}_0$ and $\mathbf{p}_1$ are initial point and the final point of the curve respectively, and τ_0 and τ_1 are their tangent vector as shown in Figure 8 (Jaklič *et al.*, 2010). In order to obtain the parameters λ_0 and λ_1, the angle equality constraint of the PH curves in Bezier form $\angle(\Delta\mathbf{b}_0,\Delta\mathbf{b}_1)=\angle(\Delta\mathbf{b}_1,\Delta\mathbf{b}_2)$ is simplified to

$$(\tau_0-\tau_1)\Delta\mathbf{b}_1=0. \tag{16}$$

with new unknowns introduced as

$$\xi_0=\frac{\lambda_0-\lambda_1}{2\|\Delta\mathbf{p}_0\|},\ \xi_1=\frac{\lambda_0+\lambda_1}{2\|\Delta\mathbf{p}_0\|},\ \text{and } \mathbf{v}=\frac{\Delta\mathbf{p}_0}{2\|\Delta\mathbf{p}_0\|}, \tag{17}$$

the equation (14) simplifies to

$$\xi_0=\frac{1}{2}\frac{(\tau_0-\tau_1)\cdot\mathbf{v}}{(1-\tau_0\cdot\tau_1)}. \tag{18}$$

Applying the constraint for the length of the edge of control polygon $\|\Delta\mathbf{b}_1\|=\sqrt{\|\Delta\mathbf{b}_0\|\|\Delta\mathbf{b}_2\|}$, a quadratic equation for ξ_1 can be written as follow (Jaklič *et al.*, 2010; Farouki and Sakkalis, 1990):

$$(1+2\boldsymbol{\tau}_0\cdot\boldsymbol{\tau}_1)\xi_1^2-2(\boldsymbol{\tau}_0+\boldsymbol{\tau}_1)\cdot\mathbf{v}\xi_1+1-(1-2\boldsymbol{\tau}_0\cdot\boldsymbol{\tau}_1)\xi_0=0. \tag{19}$$

The solutions of equation (19) are (Jaklič *et al.*, 2010)

$$\xi_1^{\pm}=\frac{1-(1-2\boldsymbol{\tau}_0\cdot\boldsymbol{\tau}_1)\xi_0^2}{(\boldsymbol{\tau}_0+\boldsymbol{\tau}_1)\cdot\mathbf{v}\pm\sqrt{((\boldsymbol{\tau}_0+\boldsymbol{\tau}_1)\cdot\mathbf{v})^2-(1+2\boldsymbol{\tau}_0\cdot\boldsymbol{\tau}_1)(1-(1-2\boldsymbol{\tau}_0\cdot\boldsymbol{\tau}_1)\xi_0^2)}} \tag{20}$$

From equation (17), (18), and (20), λ_0 and λ_1 are obtained (Jaklič *et al.*, 2010):

$$\lambda_0=\|\Delta\mathbf{p}_0\|(\xi_1^{\pm}+\xi_0),\ \ \lambda_1=\|\Delta\mathbf{p}_0\|(\xi_1^{\pm}-\xi_0). \tag{21}$$

If the angles $\phi_0=\angle(\boldsymbol{\tau},\Delta\mathbf{p}_0)$ and $\phi_0=\angle(\Delta\mathbf{p}_1,\boldsymbol{\tau}_1)$ satisfy $\phi_1+\phi_2<4\pi/3$, a unique admissible Hermite interpolating PH cubic curve exists and the curve in the Bezier form is given by (Jaklič *et al.*, 2010).

$$\lambda_0=\|\Delta\mathbf{p}_0\|(\xi_1^{+}+\xi_0),\ \ \lambda_1=\|\Delta\mathbf{p}_0\|(\xi_1^{+}-\xi_0). \tag{22}$$

For the arc-length re-parameterization, the square root of hodograph curve is rewritten as follows (Jaklič *et al.*, 2010):

$$\begin{aligned}\sqrt{\left(\frac{dx}{dt}\right)^2+\left(\frac{dy}{dt}\right)^2}&=\sqrt{\mathbf{r}'(t)\cdot\mathbf{r}'(t)}\\&=C_1B_0^2(t)+C_2B_1^2(t)+C_3B_2^2(t)\end{aligned} \tag{23}$$

where $C_1=3\lambda_0, C_2=\frac{3}{2}((\boldsymbol{\tau}_0+\boldsymbol{\tau}_1)\cdot\Delta\mathbf{p}_0-(\lambda_0+\lambda_1)(1+\boldsymbol{\tau}_0\cdot\boldsymbol{\tau}_1))$, and $C_3=3\lambda_1$. The arc-length re-parameterization that is a map from the arc-length to parameter $\Phi:[0,L]\to[0,1]$, where the length of the curve is $L=(C_1+C_2+C_3)/3$

$$t=\Phi(s)=\frac{C_1-C_2+\sqrt[3]{\frac{2}{\alpha_1}}(C_1C_3-C_2^2)-\sqrt[3]{\frac{\alpha_1}{2}}}{C_1-2C_2+C_3} \tag{24}$$

where

$$\begin{aligned}\alpha_1(s)&=\alpha_2(s)+\sqrt{4(C_1C_3-C_2^2)^3+\alpha_2^2(s)}\\ \alpha_2(s)&=(C_1-C_2)((C_1-C_2)^2+3(C_1C_3-C_2^2))\\&\quad-3(C_1-2C_2+C_3)^2s\end{aligned} \tag{25}$$

From equation (24) and (25), the control parameter can be directly calculated without numerical integrations. As a result, point information on the curve can be directly derived from an arc-length by using equation (11) and (24).

Algorithm 4 SmoothingPath (P_{pri})

```
 1: $P_{smooth} \leftarrow \varnothing$
 2: **for** $i \leftarrow N$ **to** 0 **do**
 3:   $P_{PH} \leftarrow$ PH_Cubic ($\mathbf{q}_0$, $\mathbf{q}_i$, $\Delta s$)
 4:   **if** CollisionFree($P_{PH}$) **and** Feasible($P_{PH}$) **then**
 5:     $P_{tail} = \{\mathbf{q}_i, \ldots, \mathbf{q}_n\}$
 6:     **return** $P_{smooth} \leftarrow P_{PH} \cup P_{tail}$
 7:   **endif**
 8: **end for**
 9: **if** $P_{smooth} \leftarrow \varnothing$ **then**
10:   **return** $P_{pri}$
11: **endif**
```

3.2. Connection between Current State and Primitive Path

The overall smoothing algorithm is shown in Algorithm 4. Algorithm 4 receives a primitive path $P_{pri} = \{\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_N\}$ as input, and sequentially selects from the farthest state toward the initial state $\mathbf{q}_0$ as the target state for the generation of a PH cubic curve (Algorithm 4, line 2). N represent the total number of states that compose the primitive path P_{pri}. At each iteration, the PH cubic curve P_{PH} is generated connecting the initial state $\mathbf{q}_0$ to the selected state $\mathbf{q}_i$ (Algorithm 4, line 3).The PH curve P_{PH} has the same format of a primitive path, which means P_{PH} is the sequence of a set of states. This PH curve is evaluated by checking for collision and feasibility. The feasibility of the PH curve is checked by using equation (2). If both the conditions are satisfied, the tail path P_{tail} is departed from the primitive path P_{pri} by cutting at state $\mathbf{q}_i$. Finally, the Algorithm 4 constructs and returns a smooth path P_{smooth} by connecting the PH curve P_{PH} and the tail path P_{tail} (Algorithm 4, line 4-6). In a worst case scenario such as an infeasible or unsafe path, every state $\mathbf{q}_i$ cannot be connected to the initial state $\mathbf{q}_0$. In this case, Algorithm 4 returns the primitive path (Algorithm 4, line 9-11). Although the entire path can be smoothed by connecting each state based on the PH cubic curve, the initial part of the path is only smoothed because this part of the path mainly affects control of the vehicle motion. Smoothing only the initial part of the primitive path provides efficiency in computation by reducing the collision checking and generating curves.

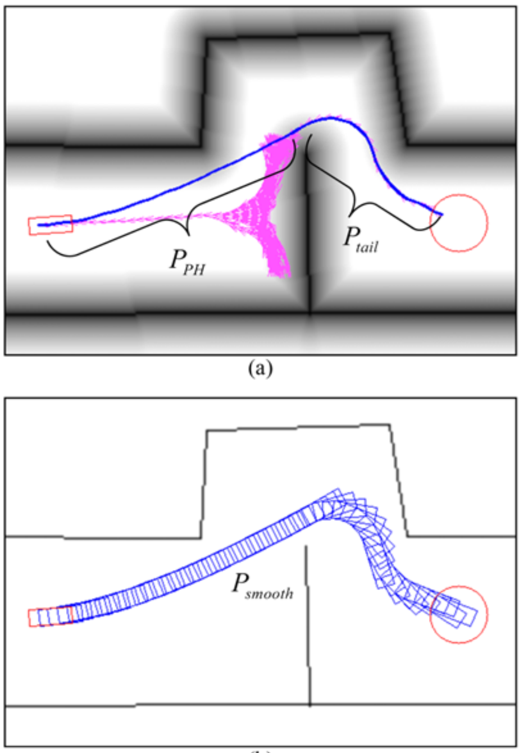

Figure 9. Smoothing based on the PH cubic curve.

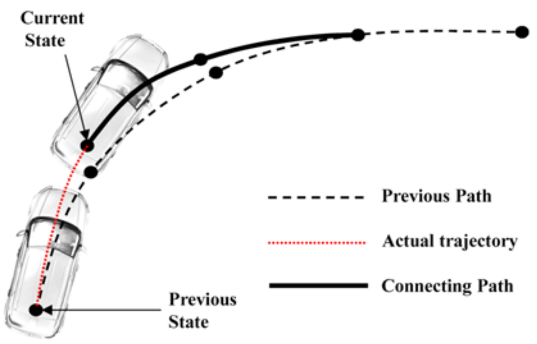

Figure 10. Connection from the current state to the previous path.

4. ONLINE RE-PLANNING

For real-time execution of path planning with limited computing resources, it is necessary to reuse information from the previous computation step. If information from the previous cycle is discarded, heavy computational work is repeated for expanding the graph and search. Furthermore, if the information is not reused and hence the path is planned independently, the path at every planning cycle is different from each other, bringing out the wavy motion.

On the other hand, if a graph from the previous planning

Algorithm 5: FindPath ($\mathbf{q}_{init}$, $\mathbf{q}_{goal}$)

```
1:  $P_{con} \leftarrow$ UpdatePreviousPath($P_{prev}$, $\mathbf{q}_{init}$)
2:  $u_{init} \leftarrow (\mathbf{q}_{init}, G)$, $u_{goal} \leftarrow (\mathbf{q}_{goal}, G)$
3:  **foreach** $\mathbf{q} \in P_{con}$ **do**
4:  $u_{new} \leftarrow (\mathbf{q}, G)$, $e_{new} \leftarrow (u, u_{new}, G)$
5:  $Q \leftarrow Q \cup \{u_{new}\}$
6:  UpdateCost(e, $u_{new}$, $G$), $u \leftarrow u_{new}$
7:  **end**
8:  **whlie** $Q \neq \varnothing$ **do**
9:  $u \leftarrow \min_{u \in Q} f(u)$
10: $Q \leftarrow Q \setminus \{u\}$
11: **if** $\|u - u_{goal}\|_2 \leq B_g$ **then**
12: **return** $P_{pri} \leftarrow$ ReconstructPath($G$, $u$)
13: **endif**
14: M $\leftarrow$ GenrateManeuver($G$, $u$)
15: AddManeuver($G$, $u$, $M$)
16: **end while**
17: $P_{smooth} \leftarrow$ SmoothingPath($P_{pri}$), $P_{prev} \leftarrow P_{smooth}$
18: **return** $P_{smooth}$
```

cycle is entirely used, additional computation is required to perform the coordinate conversion for all vertices in the graph caused by vehicle motion. For this reason, the path from the previous step is reused for generation of a new path in this approach. To reuse a previous path, the path planner maintains the previous path and performs the coordinate conversion by referencing the current vehicle state.

Although the vehicle control system can track the previous path precisely by feedback control, the tracking error for the given path could still be non-zero errors. Therefore, the previous path cannot be directly reused without a rewiring process that connects the current state and the previous path. In order to address this issue, we connect the latest current state and a point on the previous path by using the PH cubic curve as shown in Figure 10. This connecting process is applied to the closest point on the previous path, which is feasible and collision-free.

Algorithm 5 presents the entire process of the path planning algorithm. If the connected path for the closest point is infeasible or has collisions, this connected path is discarded and the next closest point on the previous path is selected to rewire to the current state (Algorithm 5, line 1).

After connecting the current state and the previous path, the connected path P_{con} is used to generating a new graph. Each state of this connected path is assigned to the vertex and the edges of the new graph (Algorithm 5, line 4). At the same time, these vertices are added to the open-list Q and the cost for each vertex and edge is updated (Algorithm 5, line 5-6).

The graph is expanded again by using the previous path information with the same process as Algorithm 1 (Algorithm 5, line 8-16). If the expanding process meets the condition of the path finding, a primitive path P_{pri} is generated by the reconstruction process (Algorithm 5, line11-12). This primitive path is smoothed and a smooth path P_{smooth} is generated (Algorithm 5, line 17). The previous P_{prev} is updated as the smooth path P_{smooth} for reuse of the next step.

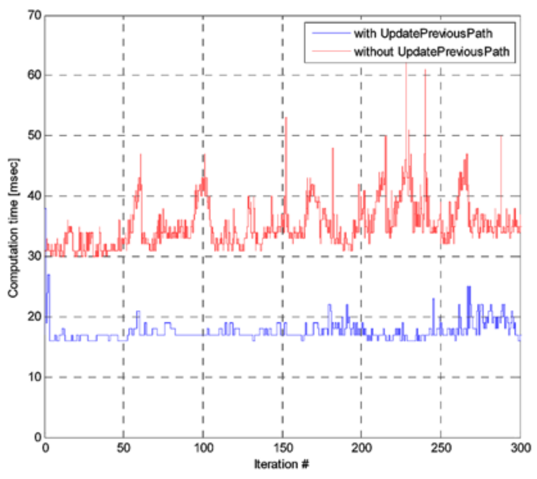

Figure 11. Comparison of the proposed path planner with and without reuse of previous path information.

This approach requires only a few evaluations for vertices and edges, while greatly reducing the number of graph expansions. If the vehicle tracks the previous path P_{prev} precisely and the perception data is not changed in the ideal case, the previous path P_{prev} can almost be reused to generate a new path. Figure 11 shows the benefit of reusing the previous path information. The graph represents computation time for path planning with and without reuse of the previous path information. As shown in Figure 11, the reuse approach can reduce both the computation time and time variation in practice. As a result, the proposed path planner can improve the computational efficiency in the real-time application.

5. SIMULATION AND EXPERIMENT

In order to evaluate the potential of the proposed path planner, we explored their performance on simulation and experimental test cases of road scenarios. In the simulation, it is compared with the most popular path planning approach: Rapidly-exploring Random Tree (RRT) (LaValle, 1998; Kuffner Jr and LaValle, 2000; LaValle and Kuffner Jr, 2001; Knispel, 2012). Since the RRT based approach can guarantee probabilistic completeness, this approach is widely used in robotics planning application (Kuffner Jr and LaValle, 2000). A detailed overview of the approaches can be obtained in (Knispel, 2012). They have been compared using the same test scenarios. In the experiment, the proposed path planning algorithm was implemented and tested on the autonomous vehicle A1. Through the experiments on real unstructured road scenarios, real-time performance of the proposed algorithm was validated.

5.1. Simulation Results

5.1.1. Scenario

To test the different planning algorithms in simulations, a set of 100 test scenarios were generated. Each scenario consists of a different initial and final condition with a situation similar to actual driving such as parking lot, congested cross road, rockfall on road, and so on. Based on the test scenarios, the proposed path planner is comparatively evaluated using four criteria such as computation time, number of nodes, final path length, and failure rate. The computation time and number of nodes are related to computational cost of an algorithm. The final path length is one of optimality criteria for a path planner. To evaluate path planning completeness, failure rate is derived from the simulation results within limits: maximum planning times (one second) and maximum number of nodes (1,000 nodes). Since the real-time capability is also important criteria, if no path was found within the limits then the test case was a failure.

5.1.2. Results

To evaluate performance of the proposed planning algorithm with previous criteria, three RRT approaches

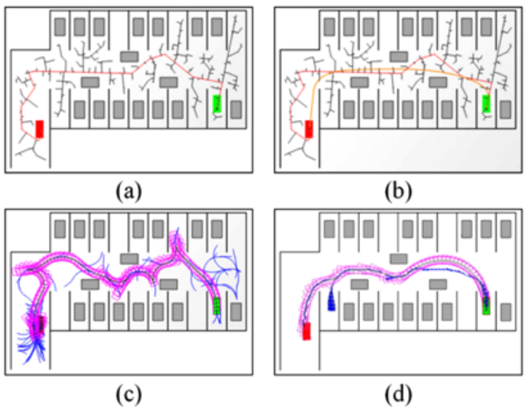

Figure 12. Example of a path planning result for parking lot scenario: (a) Basic RRT; (b) RRT smoothing; (c) RRT nonholonomic; (d) Proposed path planning.

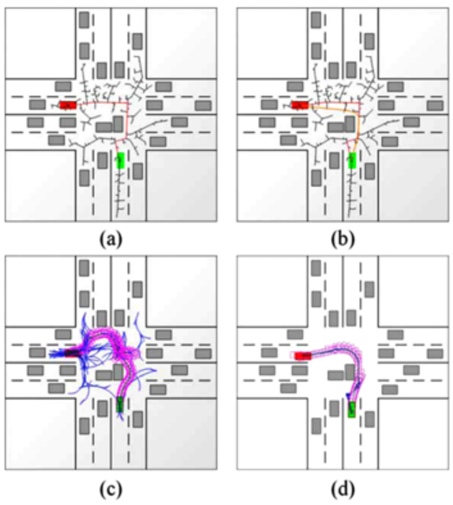

Figure 13. Example of a path planning result for congested cross road scenario: (a) Basic RRT; (b) RRT smoothing; (c) RRT nonholonomic; (d) Proposed path planning.

were also tested: Basic RRT, RRT smoothing, and RRT nonholonomic (Knispel, 2012). The RRT smoothing is based on the basic RRT algorithm. After successful finding of the path by the basic RRT, the path is then smoothly reduced by quadratic Bezier curves. The RRT nonholonomic generates nonholonomic trees which can represent sampling–based vehicle motion. By using the nonholonomic trees, the RRT nonholonomic can find a feasible path for satisfying differential constraints.

Figure 12 and Figure 13 show the path planning results of two scenario examples: parking lot, and congested cross road. In these figures, the green box and red box represent start and goal state respectively. As shown in Figure 12 and Figure 13, the path result of basic RRT are not appropriate for vehicle motions. The RRT smoothing shows improved results in respect of path feasibility, but there partially exist sharp curvature changes. Unlike the previous RRT approaches, the path of RRT nonholonomic can guarantee the path feasibility from the nonholonomic trees. However, the path length optimality cannot be guaranteed as shown in Figure 12 and Figure 13. In contrast, the proposed path planner not only satisfies kinematical feasibility, but also generates relatively optimized path in both test scenarios.

A detailed comparison of the proposed algorithm with RRT approaches is shown in Table 1. The first column contains the average computation time of each algorithm's successful cases. The second column contains average number of expanded node for finding a path. The third column contains the average length of the solution path founded by each algorithm. The final column contains failure rate in the entire test set.

In computational cost, the results show that the proposed path planner is more efficient than RRT approaches. Since the proposed path planner has the heuristic cost for a graph expansion, it can quickly find a path with less nodes in successful cases. In the final path length, the RRT smoothing shows the best performance. However, since the RRT smoothing does not consider nonholonomic constraints of a vehicle, some final paths are not feasible for the vehicle motion. In contrast, the RRT nonholonomic can consider kinematically feasibility of a path, but it is hard to guarantee the path length optimality. Unlike the RRT based approaches, the proposed path planner not only satisfies kinematically feasibility but also shows excellent path length results. In the failure rate, the basic RRT based approaches show the best path completeness. In most failure cases, the RRT based approaches find a path, but it exceeds the time limit. In contrast, the proposed planner shows a low success rate in high-complex scenarios. When there are deep local minima in the scenario, then the nodes of the proposed path planner will be drastically increased. Therefore, most failure cases of the proposed planner are

Table 1. Performance comparision of the proposed planning algorithm with RRT approaches.

Algorithm	Computation time [sec]	Number of nodes [-]	Final path length [m]	Failure rate [%]
Proposed path planner	0.074	119	65.4	12
Basic RRT	0.574	443	75.2	3
RRT smoothing	0.586	443	60.7	3
RRT nonholonomic	0.918	939	123.4	17

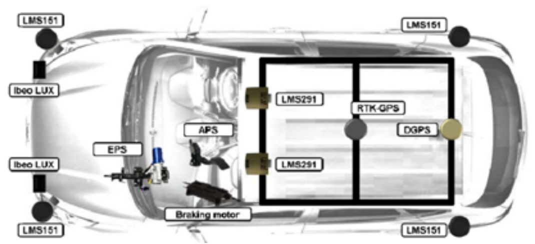

Figure 14. Sensors and actuator configurations of A1.

caused by exceeding node limitation. In addition, the proposed path planner find a path around a goal position in the failure cases, so it can finally achieve a solution path if the autonomous vehicle follows the previous path and uses replanning techniques.

As a result, the proposed path planner has excellent real-time performance, and if the complexity of path planning problem is not severe, it is more efficient than other algorithms.

5.2. Experimental Results

5.2.1. Experimental setup

The proposed path planning algorithm was implemented to the autonomous vehicle A1, which is based on a Hyundai Tucson ix platform with additional sensors and actuators as shown in Figure 14 and Table 2. To measure the location of the A1, the Real-time kinematic Global Positioning System (RTK-GPS) and the Differential Global Positioning System (DGPS) are installed on the roof of the A1. For obstacle detection, multiple laser scanners which are composed of Ibeo LUX, LMS 151, and LMS 291 are installed on the bumper and the roof of A1. To control the steering of A1, the Electric Power Steering (EPS) system of Tucson ix is used. The analog signal of Acceleration Pedal Sensor (APS) that is sent to the engine management system is emulated for acceleration control. For deceleration of the A1, the brake pedal is controlled by a braking DC motor that is mechanically connected.

Table 2. Sensor specifications of A1.

Sensor	Characteristics and configurations
Ibeo LUX	4 layer laser scanner / Horizontal 85 deg FOV with 0.125 deg angular resolution / Vertical 3.2 deg FOV with 0.8 deg angular resolution /200 m maximum range
SICK LMS 151	Single layer laser scanner / 270 deg FOV with 0.5 deg angular resolution / 50 m maximum range
SICK LMS 291	Single layer laser scanner / 100 deg FOV with 0.25 deg angular resolution / 80 m maximum range
DGPS HUACE B20	0.75 m RMS horizontal error / 12x L1 GPS channels / 10 Hz output rate
RTK-GPS ProPak-V3	GPS 1 cm real-time kinematic positions / RT-2 corrections and raw data, code positions and DGPS / 20Hz output rate

In the operation of autonomous vehicle A1, the entire driving system is designed with a 10 Hz base clock. The proposed path planer also normally runs at 10 Hz. However, the proposed path planner may take a longer time in difficult driving scenarios to generate the initial path. Once a path is generated, the next planning can be faster by reuse of the path at the previous step. For handling the obstacle map and generation of the risk map, the OpenCV library was applied (Bradski and Kaehler, 2008). To deal with the graph data structure, the Boost Graph Library (BGL), that is a library implemented using a generic programming technique, was used to efficient represent each vertex and edges (Siek *et al*., 2002; Beevers and Peng, 2003).

In order to evaluate the performance of the proposed path planning algorithm, the tests were also executed on real unstructured road scenarios: complex obstacles and construction site. The purpose of the complex obstacle scenario is to evaluate the performance of the proposed planner that finds a drivable path through randomly located obstacles. The construction site scenario aims to evaluate the ability to find a detour around the construction site. In both tests, the autonomous vehicle should iteratively find a path until it reached the goal position.

5.2.2. Results

Figure 15 shows the test results of the path planner at the

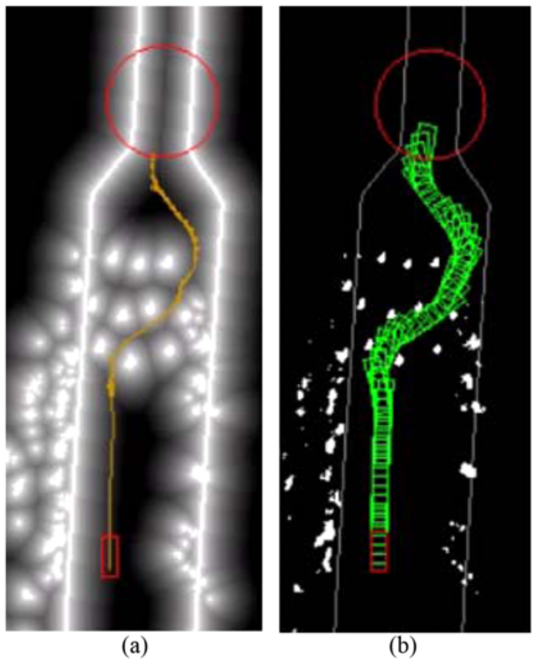

Figure 15. Path planning test at the complex obstacles.

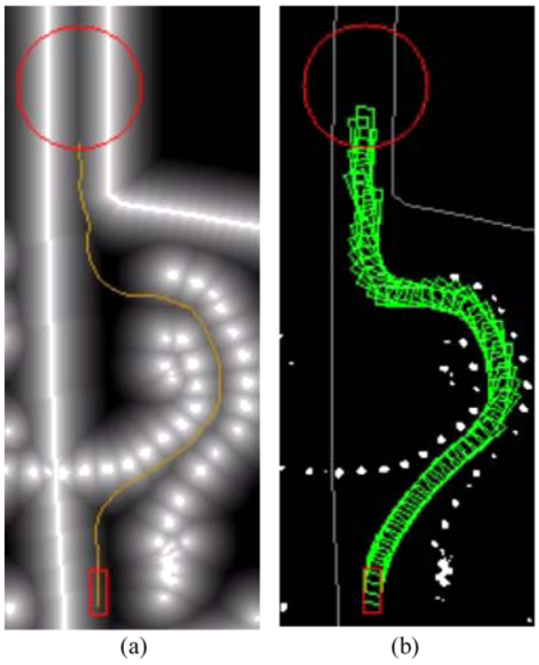

Figure 16. Path planning test at the construction site.

complex obstacles. In Figure 15 (a), the gray color represents the risk cost and the yellow color represents the graph that is generated by the path planner. The red circle is the goal area and the red box is the current vehicle position. The initial part of the graph that has no branches is made from the path at the previous step. The graph is expanded from the entrance of the obstacle region to the goal region without exhaustive expansion of the graph. Figure 15 (b) shows the generated path with the obstacle map collected from the laser scanners of the autonomous vehicle A1.

In Figure 16, a test result at the construction site is presented. Most of part of the previous path was used to expand the graph in Figure 16 (a) for efficient path planning. The initial part of the result path in Figure 16 (b) was smoothed from the graph in Figure 16 (a) by using the PH cubic curve.

The proposed path planner plays an important role in the AVC 2012 in Korea. Among several missions of AVC 2012 in Korea, the complex obstacle mission and the construction site mission require a free-form path planner for autonomous driving. This smoothed path makes it possible to drive smoothly. The logged data that were obtained during the autonomous driving of A1 in the AVC 2012 course are shown in Figure 17 and Figure 18. The gray color represents the obstacle data that were collected during the racing, and the green lines represent the road map that was drawn by using the given waypoint. The blue line indicates the generated path and the dashed red line represents the trajectory of the vehicle. In the complex obstacle mission of Figure 17, the obstacles were laid at random positions on the road. The autonomous vehicle A1 inspected the drivable area by using a structured path planner at the normal driving situation. If the structured path planner cannot find a suitable path, a free-form path planner of A1 is executed for complex path generation as shown in Figure 17 (a). In Figure 18, the construction site mission is presented. Our proposed path planner worked well to complete both the complex obstacle mission and the construction site mission of AVC 2012. A video clip about driving results is available at http://youtu.be/dzE2WReJeSM.

6. CONCLUSION

In this paper, we have presented local path planning in

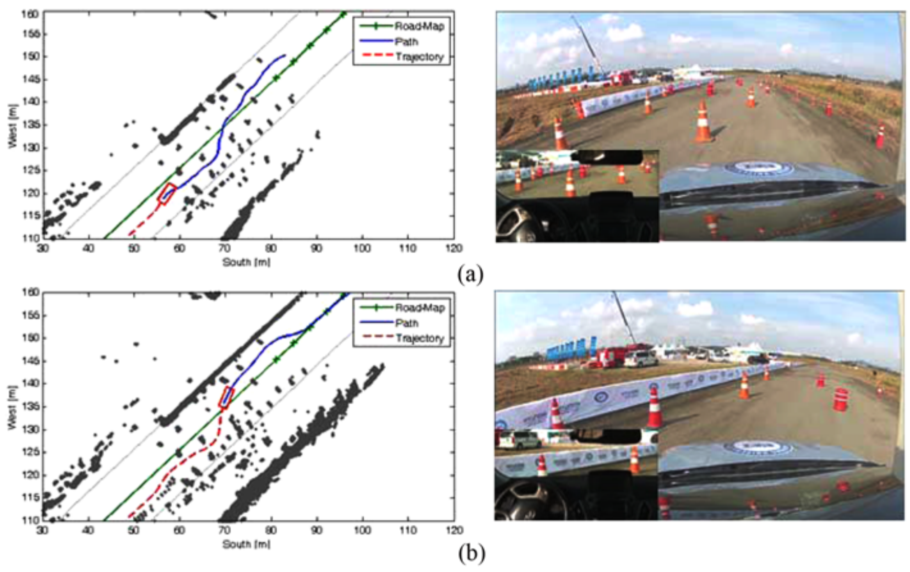

Figure 17. Autonomous driving at complex obstacles at AVC 2012.

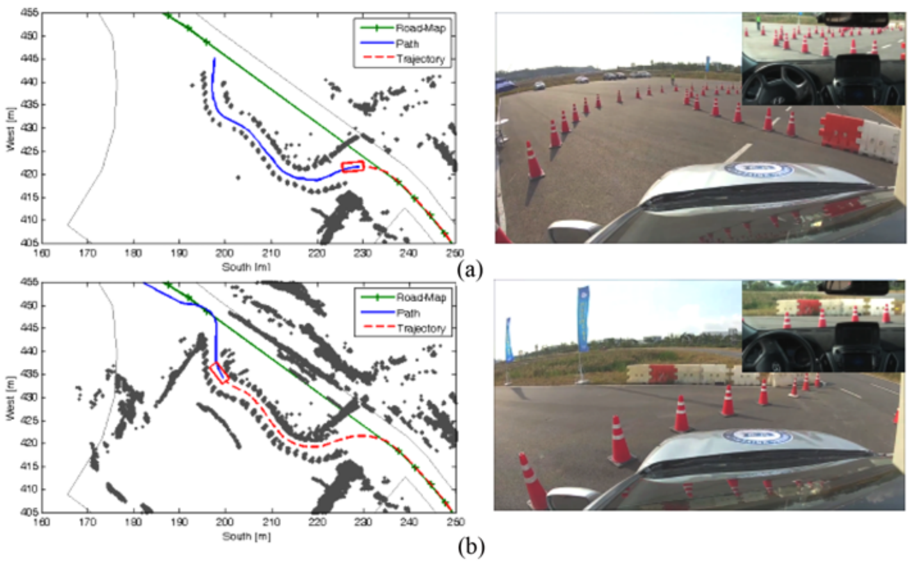

Figure 18. Autonomous driving on construction site at AVC 2012.

unstructured environments for autonomous vehicles. The proposed path planner generates a feasible path and improves the quality of the feasible path by using a Pythagorean Hodograph (PH) cubic curve. For inspection of the free space, a graph-based heuristic search is employed to produce a primitive path by discretizing and reducing the vehicle states. This primitive path satisfies the kinematic constraint, and it is collision-free and safe due to the combination of a graph satisfying the non-holonomic constraint of the vehicle motion, the risk map, and collision checking on the obstacle map. To improve the smoothness of the path, instead of applying the numerical optimization of the path, a shortcut technique is adopted to guarantee the numerical stability sacrificing the optimality of the solution for online implementation. The PH cubic curves are used to make shortcuts on the primitive path by connecting an internal state of the primitive path to the initial state, which make it possible to enhance the quality of the path satisfying a non-holonomic constraint, smoothness, and a collision-free path. The PH cubic curve provides the arc-length re-parameterization of the curve in a closed form. These properties of the PH cubic curve simplify the formulations of the maneuvering for obstacle avoidance with smooth motion in terms of implementation. In addition, the PH dos not require numerical integration to get an arc-length of curve, so it has excellent computation performance in real-time implementation. The proposed path-planning algorithms were implemented and tested successfully on the autonomous vehicle A1, which won both the 2010 and 2012 Autonomous Vehicle Competition (AVC) organized by the Hyundai Motor Group in Korea.

ACKNOWLEDGEMENT–This work was supported in part by the BK21 plus program(22A20130000045) under the Ministry of Education, the Ministry of Knowledge Economy (MKE) and Korea Institute for Advancement of Technology through the Workforce Development Program in Strategic Technology, Energy Resource R&D Program under Grant 2006ETR11P091C, the National Research Foundation of Korea, funded by the Korean government through the Ministry of Education Science and Technology, under Grant 2011-0017495, and the Industrial Strategy Technology Development Program of MKE under Grant10039673.

REFERENCES

Alves Neto, A., Macharet, D. G. and Campos, M. F. M. (2010). On the generation of trajectories for multiple UAVS in environments with obstacles. *J. Intelligent and Robotic Systems: Theory and Applications* **57**, **1–4**, 123–141.

Bacha, A., Bauman, C., Faruque, R., Fleming, M., Terwelp, C., Reinholtz, C., Hong, D., Wicks, A., Alberi, T., Anderson, D., Cacciola, S., Currier, P., Dalton, A., Farmer, J., Hurdus, J., Kimmel, S., King, P., Taylor, A., Van Covern, D. and Webster, M. (2008). Odin: Team victorTango's entry in the DARPA urban challenge. *J. Field Robotics* **25**, **8**, 467–492.

Beevers, K. and Peng, J. (2003). A* Graph Search within the BGL Framework. Boost Graph Library.

Bergenhem, C., Huang, Q., Benmimoun, A. and Robinson, T. (2010). Challenges of platooning on public motorways. *17th World Cong. Intelligent Transport Systems*.

Bishop, R. (2005). *Intelligent Vehicle Technology and Trends*. Artech House.

Bradski, G. R. and Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly.

Chu, K., Han, J., Lee, M., Kim, D., Jo, K., Oh, D.-E., Yoon, E., Gwak, M.-G., Han, K., Lee, D., Choe, B., Kim, Y., Lee, K., Huh, K. and Sunwoo, M. (2011). Development of an autonomous vehicle: A1. *Trans. Korean Society of Automotive Engineers* **19**, **4**, 146–154.

Chu, K., Lee, M. and Sunwoo, M. (2012). Local path planning for off-road autonomous driving with avoidance of static obstacles. *Intelligent Transportation Systems*, *IEEE Trans.* **13**, **4**, 1599–1616.

Dolgov, D. and Thrun, S. (2009). Autonomous driving in semi-structured environments: Mapping and planning. *Robotics and Automation*, *2009. ICRA '09. IEEE Int. Conf.*, 3407–3414.

Dolgov, D., Thrun, S., Montemerlo, M. and Diebel, J. (2010). Path planning for autonomous vehicles in unknown semi-structured environments. *Int. J. Robotics Research* **29**, **5**, 485–501.

Farouki, R. T. and Sakkalis, T. (1990). Pythagorean hodographs. *IBM J. Research and Development* **34**, **5**, 736–752.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2004). Distance Transforms of Sampled Functions. Technical Report 2004-1963, Cornell Univ. CIS.

Ferguson, D. and Stentz, A. (2005). *The Field d* Algorithm for Improved Path Planning and Replanning in Uniform and Non-uniform Cost Environments*. Robotics Institute. Carnegie Mellon University. Pittsburgh. PA, Tech. Rep. CMU-RI-TR-05-19.

Ferguson, D., Howard, T. M. and Likhachev, M. (2008). Motion planning in urban environments. *J. Field Robotics* **25**, **11–12**, 939–960.

Hart, P. E., Nilsson, N. J. and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics*, *IEEE Trans.* **4**, **2**, 100–107.

Horowitz, R. and Varaiya, P. (2000). Control design of an automated highway system. *Proc. IEEE* **88**, **7**, 913–925.

Jaklič, G., Kozak, J., Krajnc, M., Vitrih, V. and Žagar, E. (2010). On interpolation by planar cubic G2 Pythagorean-hodograph spline curves. *Mathematics of Computation* **79**, **269**, 305–326.

Jo, K., Lee, M., Kim, D., Kim, J., Jang, C., Kim, E., Kim, S., Lee, D., Kim, C., Kim, S., Huh, K. and Sunwoo, M. (2013). Overall reviews of autonomous vehicle A1 - system architecture and algorithms. *8th IFAC Symp. Intelligent Autonomous Vehicles.* Gold Coast, Australia: The Int. Federation of Automatic Control (IFAC).

Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *Int. J. Robotics Research* **30**, **7**, 846–894.

Knispel, I. L. (2012). *Advanced Robot Path Planning (RRT).* M. S. Thesis. Brno University of Technology.

Kuffner Jr, J. J. and Lavalle, S. M. (2000). RRT-connect: An efficient approach to single-query path planning. *Robotics and Automation*, *2000. Proc. ICRA'00. IEEE Int. Conf.*, 995–1001.

Kuwata, Y., Teo, J., Fiore, G., Karaman, S., Frazzoli, E. and How, J. (2009). Real-time motion planning with applications to autonomous urban driving. *IEEE Trans. Control Systems Technology* **17**, **5**, 1105.

Lavalle, S. M. (1998). Rapidly-exploring Random Trees: A New Tool for Path Planning. S. M. LaValle. TR 98-11, Computer Science Dept. Iowa State University.

Lavalle, S. M. and Kuffner Jr, J. J. (2001). Randomized kinodynamic planning. *Int. J. Robotics Research* **20**, **5**, 378–400.

Leonard, J., How, J., Teller, S., Berger, M., Campbell, S., Fiore, G., Fletcher, L., Frazzoli, E., Huang, A., Karaman, S., Koch, O., Kuwata, Y., Moore, D., Olson, E., Peters, S., Teo, J., Truax, R., Walter, M., Barrett, D., Epstein, A., Maheloni, K., Moyer, K., Jones, T., Buckley, R., Antone, M., Galejs, R., Krishnamurthy, S. and Williams, J. (2008). A perception-driven autonomous urban vehicle. *J. Field Robotics* **25**, **10**, 727–774.

Levinson, J., Askeland, J., Becker, J., Dolson, J., Held, D., Kammel, S., Kolter, J. Z., Langer, D., Pink, O., Pratt, V., Sokolsky, M., Stanek, G., Stavens, D., Teichman, A., Werling, M. and Thrun, S. (2011). Towards fully autonomous driving: Systems and algorithms. *Intelligent Vehicles Symp. (IV), Baden-Baden*, 163–168.

Likhachev, M. and Ferguson, D. (2009). Planning long dynamically feasible maneuvers for autonomous vehicles. *Int. J. Robotics Research* **28**, **8**, 933–945.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A. and Thrun, S. (2005). Anytime dynamic A*: An anytime, replanning algorithm. *Proc. Int. Conf. Automated Planning and Scheduling (ICAPS)*. 262–271.

Montemerlo, M., Becker, J., Shat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pflueger, M., Stanek, G., Stavens, D., Vogt, A. and Thrun, S. (2008). Junior: The Stanford entry in the urban challenge. *J. Field Robotics* **25**, **9**, 569–597.

Pan, J., Zhang, L. and Manocha, D. (2012). Collision-free and smooth trajectory computation in cluttered environments. *Int. J. Robotics Research* **31**, **10**, 1155–1175.

Pivtoraiko, M. and Kelly, A. (2005). Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. *Proc. 2005 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS '05)*, 3231–3237.

Pivtoraiko, M., Knepper, R. A. and Kelly, A. (2009). Differentially constrained mobile robot motion planning in state lattices. *J. Field Robotics* **26**, **3**, 308–333.

Robinson, T., Chan, E. and Coelingh, E. (2010). Operating platoons on public motorways: An introduction to the sartre platooning programme. *17th World Cong. Intelligent Transport Systems*.

Shladover, S. E. (1992). The California PATH Program of IVHS research and its approach to vehicle-highway automation. *Intelligent Vehicles '92 Symp. Proc.*, 347–352.

Siek, J., Lee, L. Q. and Lumsdaine, A. (2002). *The Boost Graph Library: User Guide and Reference Manual.* Addison-Wesley. Indianapolis.

Stein, B. (1997). A Point about Polygons. *Linux J.*

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L. E., Koelen, C., Markey, C., Rummel, C., Van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A. and Mahoney, P. (2006). Stanley: The robot that won the DARPA grand challenge. *J. Field Robotics* **23**, **9**, 661–692.

Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M. N., Dolan, J., Duggins, D., Galatali, T., Geyer, C., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T. M., Kolski, S., Kelly, A., Likhachev, M., Mcnaughton, M., Miller, N., Peterson, K., Pilnick, B., Rajkumar, R., Rybski, P., Salesky, B., Seo, Y. W., Singh, S., Snider, J., Stentz, A., Whittaker, W., Wolkowicki, Z., Ziglar, J., Bae, H., Brown, T., Demitrish, D., Litkouhi, B., Nickolaou, J., Sadekar, V., Zhang, W., Struble, J., Taylor, M., Darms, M. and Ferguson, D. (2008). Autonomous driving in urban environments: Boss and the urban challenge. *J. Field Robotics* **25**, **8**, 425–466.

Werling, M., Ziegler, J., Kammel, S. and Thrun, S. (2010). Optimal trajectory generation for dynamic street scenarios in a frenét frame. *2010 IEEE Int. Conf. Robotics and Automation*, 987–993.