PATH PLANNING AND SENSOR KNOWLEDGE STORE FOR UNMANNED GROUND VEHICLES IN URBAN AREA EVALUATED BY MULTIPLE LADARS

By

JIHYUN YOON

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2011

To my parents

ACKNOWLEDGMENTS

I would like to begin by thanking my parents for supporting me and their patience during the long process to a higher degree. They have supported my education since my elementary school and kept me motivated by their interest in the projects I worked on.

I would also like to show my appreciation to my advisor Dr. Carl D. Crane and his wife, Mrs. Sherry Crane, for their support and guidance from the first day I met them in South Korea and throughout my graduate education. Likewise, I would like to thank my committee, Dr. Warren Dixon, Dr. John Schueller, Dr. Douglas Dankel, and Dr. Eric Schwartz for their valuable comments and advice. This research and all of CIMAR's unmanned vehicle work was made possible by the support of the Air Force Research Lab at Tyndall Air Force Base in Panama City, Florida, so I would like to extend a special thanks to the staff that I was able to work with there.

Finally, I would like to thank my first teacher in CIMAR, Sanjay Solanki, and my co-workers, Eric Thorn, Nicholas Johnson, Jaesang Lee, Steve Velat, Drew Lucas, Gregory Garcia, Vishesh Vikas, Ryan Chilton, Jonathon Jeske, David Armstrong, and Shannon Ridgeway. We shared many experiences in the 2007 DARPA Urban Challenge.

TABLE OF CONTENTS

page

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AFRL	Air Force Research Lab
CIMAR	Center for Intelligent Machines and Robotics
DARPA	Defense Advanced Research Projects Agency
DUC	DARPA Urban Challenge
GPOS	Global position and orientation sensor
GPS	Global positioning system
LADAR	Laser detection and ranging
MDF	Mission data file
NFM	North finding module
PMP	Partial motion planning/planner
POMDP	Partially observable Markov decision process
PRM	Probabilistic roadmap
REDCAR	Remote detection challenge and response
RHC	Receding horizon control/controller
RNDF	Road network definition file
RRT	Rapidly-exploring random tree
SARB	Smart Arbiter
SLAM	Simultaneous localization and mapping
TG	Traversability grid
THH	Toyota Highlander Hybrid
UTM	Universal transverse Mercator

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

PATH PLANNING AND SENSOR KNOWLEDGE STORE FOR UNMANNED GROUND VEHICLES IN URBAN AREA EVALUATED BY MULTIPLE LADARS

By

Jihyun Yoon

August 2011

Chair: Carl Crane
Major: Mechanical Engineering

The autonomous driving in urban areas has more problems to be solved than the off-road driving has. The research in this dissertation proposes the new methods for detecting road boundaries, saving information of the environment, and generating path that the vehicle has to follow for unmanned ground vehicles running in urban areas.

The algorithm to detect road boundaries is implemented by several methods to assign the confidential values to the results derived from the methods. The positions of the detected boundaries are transformed to the traditional grid map with the confidential values. The traversability values of the cells corresponding to the positions are adjusted by the confidential values.

The Sensor Knowledge Store to save the information of the surrounding of the vehicle uses the Quadtree data structure. The storage is very efficient for the usage memory and easy to search and update the nodes. The information that is saved in the storage helps the vehicle to generate an optimal path.

The new path planning algorithm is based on vector method that is different method from presented method. The path planner for the special behaviors, such as

parking and n-point turn, are using Rapidly-exploring Random Tree (RRT) algorithm to find paths to perform the behaviors.

The research presented covers the theory of these new methods on an unmanned ground vehicle platform at the University of Florida's Center for Intelligent Machines and Robotics (CIMAR). Results have shown that the approaches in this dissertation are very useful for the autonomous driving in urban areas.

CHAPTER 1
INTRODUCTION

The field of robotics is developing very rapidly, especially as more and more robotic vehicles are adapted to application in the real world. These vehicles are being used to operate most everywhere manually driven vehicles have been used in the past. In agriculture, the military, industry, and home security they are being designed to take over the dirty, dangerous, and difficult aspects of manual labor. One emerging field is unmanned urban driving using car-like robots. Since there are many expected and unexpected obstacles, such as pedestrians, other cars, or rough terrain, the implementation is complex. In addition, several special behaviors, such as parking, complying with traffic signals, and lane changing should be performed using intercommunication with other vehicles. Although the implementation of unmanned vehicles is difficult and challenging, this is a research field that is receiving much attention.

Background

Unmanned vehicles of many different shapes and sizes are designed to perform a variety of tasks, which they execute with varying levels of independence. Some of the vehicles are controlled by human operators via wired or wireless input, but others can be operated without any human assistance. The latter is classified as an autonomous vehicle. Unmanned underwater vehicles are used for military purposes to monitor a protected area for new unidentified objects and by researchers to study lakes, the ocean, and the ocean floor. Unmanned aerial vehicles' military role is growing at unprecedented rates. Unmanned ground vehicles (UGVs) are operated on rough and paved terrain and spaces that are unreachable for humans.

It is the UGVs, operating without any human interaction except initiation, that are under consideration for this research. To complete a mission with this vehicle, there are thousands of necessary and complicated components that have to work together successfully. The initiation process, which is the only action that the operator performs, is to command the vehicle to reach a goal point from the initial position. The initial commands can also include an abstract mission, such as a special behavior or surveillance, for a defined area.

A level of evaluation and recognition is required by a fully autonomous vehicle. Given only a goal point during the initialization process, the robot has to localize around itself, recognize its surroundings, map a path through the surrounding terrain to reach the goal, and follow the generated path with minimum error.

One problem of robot localization is that the robot has to determine its location relative to its environment. The robot most likely will need to have at least some idea of where it is for it to be able to operate successfully. To determine its location, a robot needs knowledge of two kinds of information. First, it has predefined information gathered by the robot itself during previous runs or has that information supplied by an external source. Second, the robot obtains information about the environment through a process of evaluation of data from the raw sensors. In general, the predefined information specifies certain features that are time-invariant such as static obstacles or traffic lines. To gain this knowledge, the robot has to have single or multiple sensors, such as cameras and range laser sensors, and be able to translate data from the instruments to readable format. One option is a map describing the environment. The map can be geometric or topologic. Geometric maps describe the environment in metric

terms. Topologic maps describe the environment in term of characteristic features at specific location. Another option is that the robot learns the map of the environment while it is navigating through it, which is known as Simultaneous Localization And Mapping (SLAM)[1] and Detecting And Tracking Moving Objects (DATMO)[2]. SLAM generates an environmental map-based on sensor data, while simultaneously updating the robot's current position. Another approach is to use the Global Positioning System (GPS) from satellites coordinated with an inertial navigation system that can determine its position and motion to the robot.

All the options are created from the raw data from the sensors. The raw data must be combined to generate a representation that is useful to the robot. The most common way to construct a representation is to generate a grid map. A grid map rasterizes the environmental data from the sensors into corresponding cells and assigns a degree to each cell. Still another way is to use a vector-based representation to describe the environment. This method is closer to the raw data and describes the environment in terms of distance and angle from the robot.

To determine the control input to accomplish a goal, a proper path is necessary. This task is named path planning or motion planning. Path planning consists of the computation of the angle between the direction the vehicle is currently heading and the goal. To change the vehicle's direction, one needs kinematic and dynamic models of the vehicle and heuristic algorithms that can estimate the optimal path and generate control inputs to maneuver. There are many algorithms to find the optimal path under development. The means of the *optimal,* which can be changed depending on the

developer, are the shortest travel distance, the shortest travel time, or some other metric that differentiates one potential path from another.

Once an optimal path has been determined, the inputs must be computed and given to the actuator of the robot. Typically, the control inputs consist of steering commands to change the latitude states of the vehicle and accelerator and decelerator commands to change the longitudinal state of the vehicle. These inputs are implemented through mechanical devices or electrical signals. In robotic vehicles, steering commands are implemented by operating a motor connected to the steering column of an Ackerman-steered vehicle. Velocity changes can be implemented by the electrical signal sent from a computer that controls acceleration and deceleration, or by pushing or pulling on a brake or accelerator pedal using a motor.

This sequence of the process has to be continually supplied with current information about the new state of the vehicle and the state of its environment until the vehicle completes its task. This task could require considerable computational resources to process all the raw data and run the algorithm necessary to generate the output commands. This background section briefly described all these processes which will be described and discussed in detail later.

Focus

This proposed research has two major parts: localization and path planning. Localization for a UGV is a difficult but crucial problem for several reasons. Localization data around the vehicle are used to create the path plan with certainty about the surrounding environment to reach a goal point. There are several kinds of sensors, but this research focuses only on localization using laser range sensors. For urban area driving, the process includes detecting static and moving obstacles, terrain evaluation,

traffic line detection, and road edge detection. For localization-based on a grid map, the raw data are converted to points in global coordinate systems, the points are dropped into the corresponding cells, and then each cell is evaluated to determine its degree of traversability. When the vehicle is running at high speed, however, the cell might not have sufficient points to be evaluated. Using a vector-based evaluation about the environment using a typical grid map could aid in solving this problem. There are many approaches to localization using a vector base. In this document, the approach of the topologic map that is projected onto the typical grid map will be used. Another problem in localization is the range limit that results from the fixed size of the grid map or the sensor's hardware limit. Exceeding the limit may result in problems for the path planner, especially during high speed driving. If any obstacles or sharp turns suddenly show up, the vehicle has to quickly slow down. This action has a negative effect on the vehicle and could result in an accident or mechanical failure. This could also happen when a human is driving in a strange area. What if a human has already driven the course? Crucial information on the area where the vehicle has passed could be effectively saved in sensor data storage to help to plan a much better path. This information in storage also can assist the path planner to establish speed profiles of the vehicle when it is driving the course by viewing the course beyond the grid map's size.

Many path planning algorithms are using grid map-based searching, but sometime the methods are not appropriate for urban area driving. In addition, for special behaviors such as parking or n-point turn, the path planner based on the vector-based approach could be a better solution.

Another problem for path planning and control is that there must be consideration for the motion constraints of any hardware. This is an especially important issue for car-like robots because they are subject to a nonholonomic constraint. This means that, while a vehicle may have three degrees of freedom, it can only translate in the direction it is immediately facing. In other words, the number of control inputs into the system is fewer than the number of the degrees of freedom in the system‘s configuration space. For this reason, every time the planner generates an optimal path, it has to consider this constraint to make the vehicle track the path with minimum error.

Problem Statement

A formal description of the general UGV localization can be stated as follows: Given the raw data sent from the laser range sensors, a vehicle’s position in the global coordinate system, and a set of the vehicle position data, the introduction of an algorithm should be able to generate a final representation combined with topologic map in the format that can be sent to other components and understandable to the vehicle.

Given the position information of the vehicle, the sensor knowledge store is generated to save necessary information about the surrounding environment. To manage memory and time complexity, the Quadtree algorithm [3] is introduced.

Lastly, once the localized representation on the environment surrounding the vehicle and the stored information are provided to the path planner, the latter generates the optimal path to obtain the goal and the algorithm must be able to compute the inputs to the vehicle actuators.

Motivation

This research attempts to advance the capabilities of autonomous ground robots that have been developed at the Center for Intelligent Machine and Robotics (CIMAR) at the University of Florida and at the Air Force Research Lab (AFRL) at Tyndall Air Force Base. CIMAR and AFRL have been collaborating on projects for many years and have successfully devised automated vehicles for such applications as clearing mine fields, detecting unexploded bombs, and patrolling the perimeters of military installations. Motion planning, localization, and sensor knowledge store development are well represented in the current projects.

Over the years, many different groups of people at CIMAR have successfully automated over ten UGVs. At Tyndall Air Force Base, research has been conducted with the aim of using ground robots for military base perimeter defense as part of the Remote Detection Challenge and Response (REDCAR) project. The project concept involves three different robotic platforms as follows: mobile long range detection and assessment, agile lethal and non-lethal challenge and response, and small-scale search in areas not accessible to larger platforms. The large-scale platform integrates such autonomous navigation technologies as motion planning, environmental sensing, and modeling components developed by CIMAR with a proprietary Operator Control Unit (OCU) to achieve platform control.

The most recent autonomous vehicle research at CIMAR has focused on a large-scale autonomous vehicle for navigation in an urban environment. The vehicle, named the Urban Navigator (Figure 1-1), was designed to compete in the Urban Challenge coordinated by the Defense Advanced Research Projects Agency (DARPA) and held in Victorville, California, in November of 2007. The challenge was the most recent contest

built on the success of the previous robotic challenges in 2004 and 2005. While the previous competitions were for off-road navigation, the 2007 competition focused on autonomous driving capabilities in urban environments in the presence of other moving vehicles, including other robots. Robots had to utilize complex behaviors to demonstrate the ability to drive similar to that which happen when a human is driving a vehicle and also balance the desire for speed with the necessity of obeying all traffic rules and driving safely. Basic navigation and traffic requirements included staying in the lane, obeying speed limits, maintaining a safe following distance, passing slow moving or stopped cars, completing a U-turn, and obeying intersection rules. Requirements for advanced navigation and traffic included navigating an obstacle field, parking, merging with traffic, and defensive driving [4].

The vehicle, built off a 2006 Toyota Highlander Hybrid sport utility vehicle, was modified to provide the capabilities required by the challenge. The chassis supported drive by wire throttle and braking capabilities, although a proprietary interface board was constructed to command the throttle and brakes. Smart motors were added to automate steering and gear-shifting operations and to rotate the side laser range sensors. The sensor network, designed to meet the perception requirement of the challenge, was composed of six fixed laser range sensors, two articulated laser range sensors, four cameras, three GPS receivers, wheel encoders, and an inertial navigation system. A custom built computer rack was populated with twelve dual core computers.

The localization with laser range sensors (LADARs) used by CIMAR for the Urban Challenge was based on the grid map approach. This approach was similar to the method used in the 2004 and 2005 competitions [5]. However, since more LADARs

were added for this competition, the evaluation algorithm was required to be faster and more flexible depending on the vehicle's behavior. Basically, the LADARs provide the raw data to the computers in charge of the localization process, the data are converted to find the corresponding cell for each point cloud, and each cell is assigned a traversability value. Once the grid map representation is generated, it goes to the other components to be used.

The path planner used in this competition consisted of a receding horizon controller running a modified A* search algorithm to find the optimal path through a traversability grid sent from the localization component. The A* algorithm created a search tree through the grid map, evaluated the cost of the nodes in the tree, and used these costs heuristically for searches through the tree to find a least cost path to a predetermined goal. For a best-first search algorithm, the A* algorithm keeps track of the costs of all nodes on the tree and selects the lowest cost node for expansion [6].

The Urban Challenge along with collaborative research efforts with AFRL illustrate that there are still many research topics to be developed. Researchers at CIMAR continue to supplement existing navigation and localization capabilities. Some of the topics are proposed in this document.

One topic is enhanced localization represented by a grid map combined to a vector representation with LADARs to compensate for the weakness of using only a grid map. Another topic is the sensor knowledge store. The main functionality of this storage is to support path planning by providing information on distances further than the sensors can see. Additionally, in the case where the vehicle is getting ready to sweep an area where it has already swept and the information on the course is in storage, it

can have a predefined speed profile and optimal path by considering road curvature, max allowed speed, and static obstacle existence before starting. Finally, with the above information, a new path planner, a hybrid of previous work and a vector-based path planner, is proposed for the urban area driving project. The facilities and resources at CIMAR allow an in depth study of this claim and improve the chances of this type of technology coming to fruition.

Figure 1-1. The Urban NaviGator: Team Gator Nation's Urban Challenge entry, which was taken by CIMAR just before the competition in November, 2007.

CHAPTER 2
LITERATURE REVIEW

To compare and contrast the newly devised localization, sensor data storage, and path planning algorithm, a review of published research was conducted.

The review begins by investigating how others have approached environment representation with segmentation and classification with 3D point clouds. The issue of localization is one of the crucial features for UGVs, so several different approaches were studied. There are various sensor algorithms to date that have been used for detecting obstacles and estimation terrain.

Localization

Simultaneous Localization and Mapping (SLAM)

Many researchers have introduced SLAM for both indoor and outdoor localization problems. Originally developed by Hugh Durrant-Whyte and John J. Leonard [7], SLAM is a technique used by robots and autonomous vehicles to build a map within an unknown environment (without *a priori* knowledge) or to update a map within a known environment (with *a priori* knowledge from a given map) while at the same time keeping track of their current location [1].

Using a laser range finder along with odometry, the robot can collect the point clouds related to the vehicle position. The robot positions with point clouds are formed and odometry provides dead-reckoned transformation between two positions. The SLAM attached to an Extended Kalman Filter uses the information about the position [8].

There is another approach that introduces graph formulation to the SLAM problem. Every node of the graph represents a robot pose and the set of the laser beams taken

at this pose. Edges of the graph represent relative transformation between nodes computed from matching the laser beam sets [9].

Some researchers have shown that SLAM is not able to run well in crowded urban environments since it is very difficult to extract static and continuous landmarks. For this reason, moving objects have to be detected and filtered out using SLAM with Detection and Tracking of Moving Objects (DATMO) [10].

The EKF SLAM and GraphSLAM are on opposite ends of SLAM algorithm. There are many alternative algorithms that are under development, including Fast SLAM, UKF SLAM, and CDKF SLAM [11].

Scan Matching Localization

Some localization algorithms using laser range scanners compare a set of the points from the sensors with the set previously observed. To implement this, it should be assumed in advance that the error caused by the sensor is smaller than the one from the GPS and INS system.

Geometry matching extracted from the laser scan between the current and previous set refines the results of the GPS and INS. 2D point clouds are split into *n* segments, fitted by the least squares method, and the algorithm extracts features corresponding to each segment. This process runs with one scan line and then, if the features are successfully extracted, it finds the features that can be matched with ones in the previous scan. With the comparison, it computes rotation and translation between the two positions [12].

The features could also be considered as landmarks also and this method could be used for post-processing to build a map [13]. In a usual setting, the power poles, trees, and street lights are good examples for the landmarks since these features can

be extracted easily by searching for vertical columns that have height jumps to both their left and right neighboring columns. Given the landmarks locations previously saved, the landmarks currently extracted will be associated with given information to find the absolute position of the vehicle. Then the landmarks matching algorithm plays a crucial role in tracking the current position, given the previous position and associated landmarks. Many researchers use the Kalman Filter or a particle filter to obtain the estimate using information from the sensors and the landmarks.

Another interesting approach is to extract lines with segmentation from the laser scans [14]. In static environments with sufficient geometric features, such as was at different angles and other obstacles, point-wise scan matching can be used to determine the ego-motion of the moving horizontal laser scanner. To test robustness of the matching process, a RANdom SAmple Consensus (RANSAC) algorithm in which two sets of candidate point matches are randomly selected is approached and the position is computed.

Sometimes the mean that selects the keypoint of the point clouds by such multiple methods as segmentation, curvature clusters, and mean-shift is better for place recognition [15]. The research also proposes several ways to model the structure of the region around a keypoint location. An efficient way to solve the place recognition problem is using keypoints or feature similarities in the 2D map [16]. The *k*-nearest neighbors (*k*NN) is introduced to apply to keypoint voting–based solutions for the place recognition problem.

Monte Carlo Localization

Monte Carlo Localization (MCL) is a Monte Carlo method [17] to determine the position of a robot given a map of its environment. It is basically an implementation of the particle filter method applied to robot localization.

The key idea of the sample-based approach is to represent the *posterior belief* by a set of N weighted and random samples [18]. When the robot moves, MCL generates *N* new samples approximating the robot's position after the motion command. The sample is generated randomly from the previously computed samples with probability. The raw sensor data are incorporated into sample sets by Bayes rule in Markov localization. Since MCL uses finite sample sets, some samples are not generated close enough to the correct robot position.

This method also can be implemented with 3D point clouds collected indoors [19], although there are some drawbacks. A 3D world model is harder to build than a 2D model and needs to compute the likelihoods of many beams. So the algorithm needs to reduce the points intelligently and then compares it to a 2D map.

Another interesting research topic using this method is applying a particle filter to estimate the full state of the robot and utilizing multilevel surface maps (MLS) that allow the robot to represent vertical structures and multiple levels in the environment [20]. In this work, to estimate the pose of the robot, the probabilistic localization following the recursive Bayesian filtering is introduced. To incorporate 2D motion into the 3D map, it obtains a possible outcome of the action by applying a probabilistic model, and then adapts the motion vector to the shape of the 3D surface traversed by the robot. To adapt the motion vector, it is discretized into segments corresponding to the size of the

cells in the MLS map. After concatenating all transformed motion vector segments, a new 3D motion vector is obtained.

LADAR Data Evaluation

Segmentation and Classification

The segmentation and classification technique is a very useful process to build a map of the environment. This process works with 3D point clouds to cluster the points having different characters into the appropriate groups.

The work presented in [21] classifies each point into one of three classes: surface, linear, and scatter character. The approach has two issues: shape model computation and classification of shape features. For the shape model computation, a voxel list is introduced and each point is compared with the neighboring one and the character of the point is determined. Once the character is decided, the point is classified into its corresponding class.

Much research has introduced the distribution of a normal vector of a point and its neighbors to segment between the groups with different characters [22]. Here the normal vector means the average normal vector with neighbor points, so, to compute the normal vector, it has to find its neighbors first. There could be millions of points, so the data have to be organized or a quick searching algorithm is necessary. From the normal vectors of points, the eigenvalues are found and the points are classified into the corresponding groups by the properties of the values.

One interesting approach is using an octree structure-based split-and-merge segmentation method for organizing point clouds into clusters of 3D planes [23]. This process starts by splitting all data into eight equal sub-spaces. Then the sub-spaces are split into another 8 sub-spaces recursively until it is distributed close to a 3D best-fit

plane or less than 3 points. Then a merge process is performed with checking if the normal vectors of neighbor planes are similar to the plane. If the vectors are obviously different, the plane is segmented from the neighbor plane, so it will not be in same segmentation.

Segmentation is performed for a detection of the planes with different properties using the RANSAC-based plane model [24]. The iterative refinement of the detected RANSAC plane uses inlier points that form a densely connected set. The largest connected set is found by recursively searching the inlier point cloud from different seed points. After the segmentation, the particular 3D planes are partitioned into sets of rectangular polygons based on their support from the 3D point cloud.

Another interesting method is using the local convexity criterion with normal vectors [25]. This method stores all the 2D points on an undirected graph having nodes and edges. Then a normal vector of each point is computed using neighboring points. A crucial criterion is that if the normal vectors have approximately the same direction, two points can be considered as locally convex and be assumed to be in the same segment. This criterion allows the segmentation to be efficiently solved by a region-growing algorithm as follows: First it selects a random seed point and grows the segment until no more nodes are added, then it deletes the segment from the graph.

Covariance is also a useful method to segment 3D points [26]. Covariance analysis is often used as a starting point in the classification of 3D point clouds based on geometric properties. This is performed by determining the covariance matrix for a local neighborhood surrounding the point of interest referred to as the index point. It then proceeds by examining its eigenvalue and eigenvectors. As is known, if the point is

on a smooth surface, the variance should be close to zero. The difference in the two largest eigenvalues can represent the character of the point since they represent the variance in the principle directions on the plane. A small difference will represent an interior point, and a large difference represents a boundary point.

The K-means clustering is well adapted to laser data processing, since different feature attributes that may be geometric or textural can be used depending on the desired classes. This algorithm consists of providing a hierarchical spitting clustering to extract ground points [27]. For the clustering, mean and standard deviation of the point's height are used. It considers the cluster whose average height is minimum as the ground cluster. The K-means is implemented in a hierarchical way that splits the ground cluster into two classes in which the cluster standard deviation is higher than a predefined threshold. Until the algorithm is converged, the ground cluster is split into two classes. After the process, the point clouds are segmented into ground, off-ground, and non-determined points.

To classify urban point clouds, the shapes of the segmented feature can help with labeling [28]. The isolated points are removed and all the rest are projected into a 2D scalar image representing the height of the point cloud to find potential object locations. Then each object is segmented. Once the segments representing potential objects are found, features are extracted describing the shape of the objects. The shapes of the features are decided based on the number of points, estimated volume, average height, standard deviation in height, and the standard deviations in the two principle horizontal directions. Based on the information, the feature vector for each candidate object is classified with respect to a training set of manually labeled object locations.

Terrain Evaluation

For urban area driving, there has been much research on how to develop a real time drivable road feature. This process includes finding the road boundary of the edge, estimating the center of the traffic lane, evaluating terrain roughness, and so on. In the 2004 and 2005 competitions, the robots were required to drive off-road but in the 2007 challenge they were required to drive and stay in the traffic lane.

A minimum requirement for urban driving is distinguishing between road edge and road profile. The research shown in [29] extracts road edge, which is a curb, from the LIDAR range data. The road and road edge points are first detected in elevation data, but this should not be an absolute height. Then the points are processed by selecting the candidate regions for road segment classification. The candidates are filtered by a rule-based scheme, such as the minimal road width requirement. The line representation of the projected points onto the ground plane is identified and compared to a simple road model in an aerial view to determine whether the candidate region is a road segment with road-edges.

If the sensors can provide a sufficient number of points, the grid map-based analysis could be useful even in the urban area. The sensor that had been used in the 2007 competition by many teams was the Velodyne HDL-63E S2, a sensor that can provide enough data points to evaluate the environment at high speed. Three types of the principal component analysis (PCA), point cloud PCA, grid-based PCA, and hierarchical PCA, were used to extract the road features [30]. As mentioned above, the point clouds generate three eigenvalues to characterize the point. For the grid-based PCA, the LADAR points are dropped into a 2D grid centered around the origin of the sensor. A hierarchical PCA is developed to recursively subdivide the point clouds and

extended from a 2D grid with z height. Based on the information from PCAs, drivable and non-drivable areas are distinguished and terrain roughness is analyzed.

Urban and semi-urban roads are of irregular geometry with sharp curvatures, and corners contain signs and crossings, and traffic is characterized by lower speeds, many stops, and can be bidirectional. An important feature of these road environments is the existence of curbs on either side of the road. The detection and tracking of the curbs ahead of the vehicle using LADAR can be implemented by an Extended Kalman Filter (EKF) [31]. To extract the edge, the algorithm works on the consecutive three points of the LADAR. The dynamic model to be used for EKF is set using elementary trigonometry of the range of the points. After filter prediction, if the error between the predicted value and true value is greater than an appropriate threshold, the point is segmented.

In some cases a road does not have curbs for the boundary but must be discriminated from grass. To detect grass, a statistics model of range in grass was introduced in [32]. The measured data are pre-processed with a non-linear filter that outputs the maximum value of range within a window of contiguous samples.

The classification method also can be used to extract road features from the point clouds. For this case, the work in [33] classifies the points into only two classes: smooth terrain and 3D textured terrain such as vegetation area. This algorithm evaluates four features to characterize each point: single pixel statistics, standard deviation, spectral characteristics, and derivative statistics. Based on the evaluated features, each point is classified into vegetation or non-vegetation. After the process, the analysis of the

density of points in a conical support region placed at a data point provides an estimate of the type of point and filters out the underlying support region.

Most of the autonomous vehicles have vision sensors to detect traffic lines or lanes, but when the sensors are exposed directly to sunlight or after sunset, they could not work properly, so if the laser range sensor can detect them and determine the street type, it can make the vehicle program more robust [34]. The first step is the segmentation of the laser range data based on the distance between two consecutive scan points. This paper defines the region of interest (ROI) that may have an impact on the vehicle's future motion. The width and relative position of the lane is determined by observing the scan points on both sides of the lane within a lower range of view.

Still another interesting area of research for terrain evaluation is estimating terrain roughness using self-supervised machine learning [35]. For surface classification, this uses a square scoring matrix S of size N^2 for N laser points. Comparison of feature r with feature c yields a score to the matrix. The w largest elements are extracted and accumulated in ascending order to generate the total score based on the area where the real wheels will pass in future. The total score is the output of the classifier, which means the parts with a total score greater than a predetermined threshold are classified as rough terrain and the parts less than the threshold are assumed to be smooth. The most interesting part is self-learning the parameters that are used to give the score. When the vehicle traverses the area that already has a corresponding score, the roughness coefficient is calculated. The objective function for learning is $Tp - \lambda Fp$, where Tp and Fp are the true and false positive classification rates, respectively. A true positive occurs when a given patch of road has ruggedness coefficient that exceeds a

user-selected threshold and it is important to minimize false positives. This approach is very useful to improve the terrain evaluation capability of the vehicle.

Another terrain character that has to be carefully determined during driving is negative obstacles such as holes, ditches, or cliffs. This problem can be easily solved relative to the other problems [36]. Like other solutions, this starts by data accumulation and terrain classification. Then all the clouds belong to a voxel grid, which is a 3D map, and the map is projected onto a 2D grid map. Usually negative obstacles occur only in regions where there is a transition from visible to occluded voxels and their positions are represented in the 2D grid map.

Obstacle Detection

The algorithm for obstacle detection is a crucial part to help the autonomous vehicles drive safely without colliding with any static or moving obstacles. Many researchers implementing methods to detect static obstacles have had successful results. However, detecting and tracking moving obstacles, which was one of the challenges in the 2007 competition, and feature extraction still remain considerable challenges.

A single planar scanner cannot be relied upon to distinguish between obstacles and the terrain changing to uphill. For this reason, many autonomous vehicles are equipped with multiple scanners to see the front of the vehicles or by a 3D laser scanner that can see far. One approach is to mount multiple sensors at different height on the vehicle [37]. If two sensors of approximately the same range report different heights, it could be an obstacle, otherwise it is assumed to be an uphill or non-flat road. The research uses a 2D lookup table with 1 meter resolution to store segmentations created by the obstacle sensor and the table can be used to construct the algorithm of

O(1) time complexity to find segmentation nearest a point. Once any obstacle is found, it is dropped into the appropriate row and column of the table. The main goal of this clustering is to track objects over time. At each time step, the new objects are associated with a similar group from the previous time step. The bounding boxes for two associated groups are compared and compute the velocity of the object. This research group has used the Velodyne sensor so it can see at least two corners of objects to track the objects well.

Most of the object-tracking algorithms start by grouping the point clouds. The segmentation process can provide the polygonal model of the environment and can be implemented by some algorithm derived from image processing techniques used for edge detection [38]. The Ramer algorithm [39] is used to segment whole one-scan points. From this step, all data points that cannot be included in any segment are removed. All generated segments are connected to those closest and analyzed individually based on length, number of vertices, and orientation with respect to the sensor. Then, the segments are split again based on the distance from their neighbors and if two segments share the same point, they belong to the same object. Finally, each segment is grouped into predefined objects of classes, such as car, motorcycle, truck, or undefined.

There is an obvious limit to use only a laser scanner to detect and track the objects. For this reason many researches are trying to fuse multiple sensors, such as laser scanner, vision sensor, radar, and so on. Usually, laser scanners are used with vision sensors for detection and tracking [40]. A 3D vision sensor and a 2D laser scanner can estimate an ensemble of measurement positions from the real world. Two

measurement points are assigned to the same obstacle if their Euclidean distance is below a predetermined threshold. Tracking is performed using a Kalman Filter that is based on a simple model in terms of the obstacles' position, speed, acceleration, height, width, and length. Both sensors have advantages and disadvantages, so by properly fusing these two, a good result can be expected.

Most research uses the range data returned from the laser scanners to detect and track objects. However, many sensors can also provide the intensity, which may be called reflectance of the returned signal of every beam. The reflectance provides more information about the obstacles or vertical surfaces. Using this property, a model of the laser reflectance signal is proposed [41]. The model uses actual surface reflectance, angle of the beam with the surface, and the range. The returned intensities are different for different objects, so obstacles can be detected by these differences.

Another useful area of research for autonomous vehicles is estimation and prediction of collision risk [42]. A Bayesian Occupancy Filter (BOF) operates with a 2D grid representing the environment. Each cell contains a probability of the cell occupancy. Given an observation, the BOF updates the estimate of the each cell. Fast Clustering Tracking (FCT) takes two inputs, the grid estimated by the BOF and the prediction of the tracker that provides a region of interest for each object being tracked. If there is more than one region of interest in the tracked objects which are overlapped, they extract the cluster which is associated with multiple targets. Then, the local SLAM problem is solved by dividing the laser points into stationary and moving sets. Given the occupancy grid map and the current state of the vehicle, the laser impacts generated

from stationary or moving objects can be separated using a threshold of occupancy probability.

For driving in an urban area, other important objects that must be detected are pedestrians. Usually pedestrians are in already cluttered areas, so it is considerably difficult to detect and track them. The means to do that is to use multiple sensors, such as vision sensors and laser scanners [43]. This research proposes lidar-based detection, tracking, and classification similar to the research mentioned previously and also uses a set of Haar-Like features to extract information from any given image from the vision sensor. Then this performs a core process that is combines the objects classified by two sensors by a sum rule. The sum decision rule depends on the prior probability of occurrence of each class and the posterior probabilities from the respective classifiers.

Sensor Knowledge Store

There is much information obtained from various sensors: height of the terrain, obstacle existence, terrain roughness, road curvature, road boundary, and so on. The Sensor knowledge store (sensor information database) was developed to represent the environment evaluated by such sensors as vision, radar, and laser scanners. With this information, the vehicles can plan a path to get to a goal and determine the driving constraints in advance.

Triangulated Irregular Network

One of the reasons to use sensor data storage is to reduce the amount of data. Collecting and saving all point clouds are usually inevitable so to increase the efficiency of the algorithm and reduce computation time the storage is introduced. One method to do this is Triangulated Irregular Network, TIN, algorithm [44].

The TIN was originally devised by Randolph Franklin at Simon Fraser University in 1973 [45]. It is a data structure used in a geographic information system for the vector-based representation of a surface.

TIN is a very useful idea to represent a surface, but not the best for the storage because it is “irregular”, so it takes considerable time to find a triangle corresponding to the data and update it. Consequently, it was proposed to map the TIN onto a static grid map [46].

R-trees

R-trees are tree data structures that are applied to the spatial access method to index multi-dimensional information [47]. This splits space with hierarchically nested, and possibly overlapping, minimum bounding rectangles.

Currently, the laser sensors can provide very dense point clouds, so to store, load, and visualize them, R-tree is one of the efficient methods [48]. In this research, $4 * 10^{18}$ data points can be stored in Oracle, which partitions the points into fixed size blocks and stores the blocks as rows in a separate block. Loading and visualization steps can be processed in a fixed block similar to the storing process.

The most useful aspect of the tree is indexing 3D data. The approach shown in [49] has used a V-reactive tree, which is based on combining R-trees with an importance value. The V-reactive tree is a 4D R-tree optimized for 3D visualization.

Quadtree

A Quadtree is a tree data structure in which each internal node has exactly four children. It is used to partition a 2D space by recursively subdividing it into four quadrants or regions. This is firstly discussed by Raphael Finkel and J.L. Bentley in

1974 [50]. Also, the research in this document used this Quadtree for the sensor knowledge store.

One feature of a global information system is generation of the potentially large amounts of information. The variety of potential data types is also large and extremely diverse, including numerical, categorical, or vector-valued data. For these reasons, the research in [51] had five proposed criteria to be satisfied. First, a hierarchical data structure is required. Second, a regular hierarchy is more efficient that an irregular one. Third, it should be easy to convert from existing structures used for a geographic information system and back again. Fourth, the system should have a clear and simple coordination to the currently system of latitude and longitude. Lastly, the purposes of a geographic information system are primarily for planning, analysis, and inventorying of geographic phenomena.

Generation of a large-scale digital elevation model (DEM) often incorporates the Quadtree algorithm. One report noted that DEM is characterized by its combination of adaptive Quadtree-based processing with common iterative interpolation and filtering methods [52]. These filtered out non-ground data points are based on the Quadtree iterative algorithm and segmentation algorithm.

Planning and Control

Two individual tasks will be introduced: a geometric path or time varying trajectory of the vehicle's motion and control task to follow the generated path. The path should be optimized for travel time, distance, or minimum collision potential. Sometimes, all of the parameters should be optimized. The second task considers the path as an input and generates input to control steering and change vehicle speed. Many researchers have tried to find a viable solution for dealing with the nonholonomic and other

constraints of UGVs. Although many methods were provided, there has been a tradeoff between stability, robustness, and performance.

Replanning Algorithm

Some path planning algorithms for autonomous robots are run over and over again at a high rate. However, several types of algorithms make use of the previous solution trajectory from the start point to the goal location and only make minor adjustments to the trajectory when there is any change in the environment on the path.

Once a vehicle has recognized the environment successfully, it starts generating a plan to get to a goal. The work in [53] determined the path based on the graph data structure, which consists of a set of nodes and edges. To construct the graph for the plan, traversability rays in the visible range on the cost map are emitted at different angles from the location of the vehicle. Then a cost and Exploration Bias(EB) are assigned to each cell, where the EB is zero unless the edge that connects the node to the robot collides with an obstacle. Finally, if any edge intersects another edge, they are merged and translated in different shapes. After the graph is generated, the algorithm finds a path, which is the object of this research. To do this, the algorithm needs to select an interim goal based on a heuristic function. The next step is to plan a path from the current location to the interim goal by way of the shortest path. The vehicle is performing this process over and over again until it gets to the goal point.

Sometimes the information about the environment is imperfect or incomplete. As a result, a path generated with that information may be invalid or not optimal. For this reason, it is crucial that the algorithm must be able to acquire new information. While many algorithms are providing a solution to the problem by replanning, the algorithm may not have enough time to compute the best solution, so it needs to settle for using a

suboptimal solution. One of the solutions proposed for this problem is known as the anytime algorithm [54] that starts by computing an initial, potentially highly suboptimal solution, and then improves this solution as time allows. A* based anytime algorithms show that the heuristic values used by A* provides substantial speed-ups at the cost of solution optimality.

Many incremental algorithms, such as A* and D*, have been used in robotics for mobile robot navigation in unknown or dynamic environments. However, many of the researchers recognized the approach has limitations. For example, the cost per unit of traverse given to each cell is not continuous, which means there is difficulty navigating the respective area of the environment. For this reason, a researcher devised the Field D* algorithm [55]. This algorithm is also an incremental replanning algorithm, but incorporates interpolated path cost. Field D* is an extension of classical grid-based planners that uses linear interpolation to produce more natural paths with less cost through grids.

The path planning can be decomposed into two levels: local and global. The local planning is to avoid obstacles while driving and react to data from sensors as quickly as possible. Methods that use a uniform-sized grid map must allocate memory for regions that may never be traversed or do not contain any obstacles. To compensate for this drawback, the framed Quadtree data structure has been suggested [56]. This data structure has been used to extend a path planner on the uniform grid cells. A regular Quadtree can provide efficient memory usage to the path planner, but paths generated by Quadtree are suboptimal because it is constrained to segments between the centers of the cells. To solve this constraint, it was modified in order that cells of the highest

resolution are added around the perimeter of each Quadtree region. However, there is still a problem in this algorithm in that in a congested area it needs more memory than a regular grid map.

Probabilistic RoadMap

The probabilistic RoadMap (PRM) was developed by Svestka and Overmars for single car-like robots in a static environment [57]. The basic idea of PRM is to take random samples from the configuration space of the vehicles, check them for whether or not they are in the free space, and use a local planner to attempt to connect these configurations to other nearby configurations. For this, a graph search algorithm is applied to determine a path between the vehicle and goal configuration. The RPM planner consists of two steps: a construction phase and a query phase. In the construction phase, the algorithm connects a configuration with its neighbors to make a complete graph; in the query phase, the path is obtained from the graph using Dijkstra's shortest path algorithm.

The research reported in [58] plans the path in high dimensional configuration spaces. During the construction phase, the roadmap is constructed in a probabilistic way for a given scene. The roadmap is an undirected graph $R = (N, E)$. The nodes in N are a set of configurations of the vehicle appropriately chosen over the free configuration space (C-space). An edge (a, b) corresponds to a feasible path connecting the configurations a and b. This path is computed by a local path planner. Since the computation of the path can be completed very quickly, it is not stored in the roadmap. In the query phase, the roadmap is used to solve individual path planning problems in the input scene. Given a start configuration s and a goal configuration g, the method first tries to connect s and g to two nodes a and b in N. If it is successful, it

searches *R* for a sequence of edges in *E* connecting *a* to *b*. Finally, it transforms this sequence into a feasible path for the robot by recomputing the corresponding local paths and concatenating them.

Another important aspect is to analyze the probability of failure of PRM as a function of all the relevant parameters [59]. When pairs of point *a* and *b* are connected by including the upper bound of the failure probability, they can be connected by some path that keeps them uniformly away from the obstacles. The idea is to cover the path with balls overlapping to a certain degree. The idea mentioned uses only the minimum distance from the obstacle, but if this is achieved rarely, the distance should be varied. This research shows how to implement the PRM with a simple collision free algorithm.

There is a new PRM algorithm for single and multiple query path planning. For the algorithm, PRM does not build a roadmap of feasible paths, but rather a roadmap of paths assumed to be feasible [60]. The concept is termed, Lazy PRM, which lazily evaluates the feasibility of the roadmap as planning queries are processed. The first step of the algorithm is to build an initial roadmap with a vehicle node and a goal node. The next step is the selection of the neighbor to be connected with a node in the roadmap. Then it finds the shortest path between the vehicle and the goal node. If the path is found, it checks if the path is free from obstacles. Otherwise, it reports failure or starts the node enhancement step to add more nodes to the roadmap and search again. At the node enhancement step, the algorithm generates a set of new nodes, inserts them into the roadmap, and selects neighbors in the same way as when the roadmap was initially built. To add the new nodes, it selects a number of points, called seeds, in the roadmap, and then randomly distributes a new point close to each of them. When

the planner has found a collision-free path, it terminates and reports the path. For the laziness, information about which nodes and edges have been checked for collision is stored in the roadmap, and as long as the obstacles field remains the same, it uses the same roadmap when processing subsequent queries. As several queries are processed, more and more of the roadmap will be explored, and the planner will eventually find paths via nodes and edges that have already have been checked for collision. This stored information makes the planner efficient.

Potential Fields

Another approach was proposed for robot path planning that consists of incrementally building a graph connecting the local minima of a potential field, which was developed by Oussama Khatib [61], defined in the robot's configuration space and concurrently searching this graph until a goal configuration is attained [62]. This was an efficient approach that did not require any pre-computation steps to be performed over the potential field function. The process to construct numerical potential fields in the configuration space of a robot consists of two major steps. First, potential fields, called W-potentials, are computed in the robot workspace. Each W-potential applies to a selected point in the robot, called a control point, and pulls this point toward its goal position by a function named the arbitration function. The next step is to construct a "good" C-potential in the configuration space. The path search algorithm only considers the neighbors of a configuration that are collision-free, so there may be two types of local minima: the natural minima (where the gradient is zero) and the minima located at the boundary of free configuration space.

In the potential field approach, obstacles are assumed to carry electric charges, and the resulting scalar potential field is used to represent the free space. This method

also can be used for local and global planners [63]. In this research, the obstacles can be represented by specifying their surface or volumes and if a potential field representation of the free space is used, then the valleys of potential minima (MPV) define the Voronoi diagram. The global planner in [63] selects the shortest path from the MPV graph between the start and the goal nodes with the minimum heuristic estimate of the chance of collision. Once the global path is designated, the local planner starts by analyzing the potential field in two ways. First, when the initial path and orientations involve potential collisions, the local planner modifies the robot's configuration to minimize that potential and attempts to avoid the collision. Second, although the path and orientation are free of collisions, they may not be optimal in the sense of the shortest path and the minimum changes in orientation. The local planner uses the potential field as a penalty function in a numerical algorithm that optimized the path length and orientation change. Then, if a solution is found, it will find further optimized solutions with the numerical algorithm.

Other Methods

While previous sections introduced several approaches to perform path planning for robots, many other interesting approaches still remain for the modifying path planning algorithm. The research in [64] shows the path planning algorithm using the Vector Field Histogram (VFH) method. It employs a two-stage data reduction technique and three levels of data representation. The highest level holds the detailed description of the robot's environment and this level is expressed with a 2S Cartesian histogram grid that is updated by range sensors. At the second level, a 1D polar histogram is constructed around the robot's location. The third level of data is the output of the VFH algorithm with data composed of the path for the drive and steer controllers of the robot.

The first step of the data reduction is to map a histogram at the first level onto the polar histogram at the second level. The second data-reduction stage computes the required steering angle. The robot can determine the free sector, steering angle, and speed command with the histogram grid and polar histogram. Although this method does not generate a path, but it finds the traversable area and computes input commands.

The work described in [65] is based on the Grid Distance Field (GDF), which is based on the W-potential in [62]. The GDF has several features. First, field strength changes linearly with path length from the goal. Second obstacles are avoided by virtue of not falling on the trajectory path created by the gradient. Next, the linear gradient is generated by a simple and fast algorithm that is linear based on the number of grid cells. Last, the resulting paths do not have smoothly varying trajectories. The path generation has two steps in this case. The GDF is computed and then a sequence of local waypoints is extracted from the GDF.

The problem of finding the shortest path under the curvature constraint was first introduces in the pioneering work of Dubins [66]. In robotics, a Dubins car refers to a vehicle with a minimum radius that only moves forward. Using this car, the work proposed in [67] determines a cost function, then uses it to compute individual trajectories. To do this, they introduced the Hamilton-Jacobi formulation (HJ equation) for path planning. The optimal paths from an initial pose can be computed by tracing characteristic curves defined by the HJ equation back to the goal position. In addition, the Dubins car concept is being applied to many robotics path planning areas.

The last useful and simple method is using a Bezier curve, a parametric curve frequently used in computer graphics that can be scaled indefinitely. Thus the curve is

very useful to generate a smooth path. In the paper that proposes a way to use this approach, the most important part is the selection of control points [68]. They used a 4^{th} order Bezier curve, so it needs five control points to generate a curve. Usually, the first point and the last point for the curve are the vehicle's initial position and goal position, respectively. Control points are selected that smoothly lie in the direction of the line connecting the start point and goal point. If there is any obstacle on the generated path, the vehicle needs to generate a new one with a continuous curvature. In this case, a new intermediate goal point is selected to find another path to avoid the obstacles. After that, it generates two curves. The first curve uses the vehicle's position and the intermediate goal; and the second one uses the intermediate goal and the original goal. Then the algorithm connects the two curves to generate a complete path to avoid the obstacles. The path planner using the Bezier curve is very simple and can work well, but the selection of the control points is very important.

Occupancy Grid

Mobile robots are often designed to operate in environments that are unstructured and unknown. In these cases, the system has no *a priori* model or knowledge of its surroundings. Occupancy grid structures offer a means for a robot to map and rationalize this unknown space. Each cell in the array contains a probability estimate that identifies whether the space is occupied or empty. Occupancy grids are mostly used to quickly check if there is any obstacle in some position. There are several ways to create the grid.

The research shown in [69] has proposed the two ways to build the grid: Least Median Squares(LmedS) and gradients. The LmedS method uses a non-linear optimization method to find the dominant plane in the point clouds and then uses this

plane to build the occupancy grid based on the distance of each point in the cloud from the dominant plane. The gradient based method relies on the idea that local gradients in the point cloud data can be used to determine whether a particular cell is occupied by an obstacle. The final occupancy grid is generated by fusing two maps built using the two methods. Collision checking is performed by mapping the vehicle's shape onto a further occupancy grid and recursively transforming this shape along the predicted trajectory.

The occupancy grid also can be used with probabilistic Velocity Obstacles (VOs) for dynamic obstacle avoidance [70]. The researchers suggested three steps to avoid obstacles. First is the cell-to-cell approach. Each cell is considered occupied or non-occupied, the velocity of the objects is already known, and the grid is relative to the robot. The specified collision search space is reduced to the velocities reachable within the next time step with dynamic and kinematic constraints as a maximum acceleration value with a maximum and minimum velocity in each direction. Second is to compute the probability of collision. Each cell also stores a probabilistic estimation of its state. The last step is to choose the control input. The occupancy grid provided from the previous two steps gives a set of control inputs to the vehicle.

The obstacles and the vehicles usually cannot be considered as invariant disk shapes. Assuming the simple model of a rigid robot that is restricted to translation, the configuration space obstacles can be expressed in terms of the Minkowski sum of the robot and obstacle shape [71]. This approach decomposes the vehicle shape into disks and does the Minkowski sum. Then the sum is decomposed into the axis aligned rectangles.

Control Strategies

The control task for an UGV involves the regulation of the vehicle to a predetermined motion command. The goal is to minimize such errors as heading errors or lateral errors between the vehicle's state and a desired state. This is implemented by commanding the vehicle plant inputs in a deliberate manner that is often specified by a mathematically defined function, or procedure. Many control methods have been developed for this purpose.

The research in [72] provides a number of examples of the control input for steering. They divide tracking methods into three major groups: geometric tracking, tracking with a kinematic model, and tracking with a dynamic model. A geometric vehicle model only needs the turning radius and wheel base distance. The pure pursuit, which is the method of geometric tracking, is the most common approach to the path tracking problem for mobile robots. It consists of geometrically calculating the curvature of a circular arc that connects the rear axle to a goal point on the path ahead of the vehicle. The goal point is determined from a look-ahead distance. Another geometric tracking method is the Stanley method which is used by Stanford University's autonomous vehicle entry in the DARPA Grand Challenge. This method is a nonlinear feedback function of the cross track error measured from the center of the front axle to the nearest path point. A heading error is computed using the heading of the vehicle and the heading of the path at the nearest point. The resulting steering is a function in terms of the two errors. The kinematic bicycle model is a common approximation used for robot motion planning, simple vehicle analysis, and deriving intuitive control laws. The wheels are assumed to have no lateral slip and only the front wheel is steerable. The two previous models ignored the vehicle dynamics, but that ignorance had a

negative impact on tracking performance as speeds were increased and path curvature varies.

To track the trajectory, the vehicle needs to obtain a reference steering angle and a reference speed for the next time interval to get back to the reference path ahead from a current deviated position. The path tracking can be separated in two ways: global position feedback and separate steering and driving control [73]. If the path to be tracked is specified in Cartesian coordinates, global position feedback computes the steering angle with the position error and heading error. Since the guide point is on the center of the rear axle, steering and driving reference inputs are computed from the given path and vehicle speed, respectively in the geometric model.

The extension of [73] had been proposed in [74]. The path tracking problem is approached first by formulating a control problem with the consideration of characteristics of conventionally steered vehicles, which uses three strategies. The geometric path tracking pursues “time history of position,” reducing the second problem to a steering problem. By locating the guide point on the center of the rear axle, the path determines only steering motion for tracking. The last strategy is nested feedback control loops. The steering angle can be computed by the steering planner, the computed angle goes to the steering controller, and the controller will operate the steering by comparing the computed angle with the real current steering. In a similar process, the vehicle speed can be computed.

Model predictive control is an advanced technique often used to solve optimization problems under certain constraints. The research shown in [75] presented a result

where a wheeled mobile robot was regulated to an arbitrary set-point using a receding horizon controller. They demonstrated that the vehicle stability could be achieved.

CHAPTER 3
THEORETICAL APPROACH

Environment evaluation and path planning are challenging tasks for autonomous vehicles. This dissertation presents novel approaches to these tasks by examining the efficient storage of sensor information. This chapter is divided into three functionality components: sensor evaluation, sensor storage, and path planning and control strategy.

First, a component that is composed of multiple LADAR sensors, called the Terrain Smart Sensor (TSS), evaluates the surroundings of the vehicle and provides that information to the other components. In the case of urban area driving, one of the most important aspects is to find road boundaries that the vehicle should not traverse. There are many ideas to use properties of the LADAR beams and road boundaries. The information sent from the TSS can be used immediately by other components and also be saved to help the path planner when the vehicle drives in the same area in the future. The second section is about the sensor knowledge store component. There are many ways to save the information, but the algorithm needs to consider efficient memory management, since how far the vehicle will run is unknown. This chapter introduces the Quadtree, a data structure algorithm, for sensor knowledge store and how to save, update, and send the necessary information is also presented. Last, the path planner based on the vector method is described. The vector-based planner works depending on the vehicle's behavior. There are several behaviors for the vehicle, roadway navigation that is the basic and most frequent behavior, n-point turn, and parking. The planner for each behavior runs using different approaches such as the Bezier curve and rapidly-exploring random tree (RRT). This chapter explains these approaches generally

and the next chapter applies the description to implementation on an unmanned ground vehicle.

LADAR Terrain Evaluation

Traversability Grid Map

The surrounding environment of an outdoor autonomous vehicle is highly unstructured and dynamic. The environment consists of natural obstacles that include positive obstacles such as trees and rocks, negative obstacles such as cliffs and ditches, and road boundaries such as curbs, traffic lines, and intersections. Other obstacles can include vehicles and pedestrians. The classic method for terrain evaluation uses a 2D Traversability Grid representation (Figure 3-1) that is always oriented in a north-east direction with specific number of rows and columns, specific size, and specific resolution. Each cell of the grid map has a traversability value assigned by evaluating the points collected in the cell. The lower value represents the lower traversable probability and vice versa. The representation is not only able to distinguish highly non-traversable obstacles but also able to notice differences in the terrain to the vehicle, which helps the vehicle to avoid the obstacles and to adjust its speed to pass through rough terrain.

The procedure for this grid map is as follows:

1. Transform the distance data reported from the each LADAR sensor to the global coordinate system using yaw, pitch, and roll angles of the vehicle reported from a component equipped with a GPS and an NFM.
2. Assign each transformed point into the cell with corresponding row and column.
3. Evaluate each cell by mean height, plane slope, etc. of points assigned into the cell.
4. Determine the traversability value using the predefined properties.

The circular buffer implementation of the Traversability Grid is very efficient in updating the grid to the new position as the vehicle moves. The main advantage of using a circular buffer in place of a 2D array is that, for a 2D array, as the vehicle moves, for every new position of the grid, data from the cells in the old grid are copied into the corresponding cell indices in the new grid, but an expensive computing operation of copying the grid data every time can be avoided by using a circular buffer. Figure 3-2 shows the change in the grid position due to the movement of the vehicle. As shown in the figure, all the data corresponding to the overlapping cells in the two grids have to be copied from the old grid position to the new grid position for a 2D array representation. The circular buffer implementation of the Traversability Grid avoids this computationally expensive operation by storing the data in the grid as a 1D array of size equal to the number of cells in the grid. Figure 3-3 shows this procedure.

Mapping in Urban Areas

The aforementioned classic grid map approach had worked well for the 2005 DARPA Grand Challenge that was for off-road competition, but it is deficient for driving in an urban area. One problem is how to detect low height road boundaries such as, curbs. Usually, the difference between the mean height of the cell with a curb boundary and road cells is not large enough to be distinguished. For this reason, the component needs extra functionality, called the vector-based edge finder, to detect those road boundary features. Figure 3-4 shows the TSS component with the Edge Finder functionality. The procedure of this functionality is as follows:

1. The data points from the LADARs that can evaluate the terrain are transformed into a global coordinate system.
2. The data points are separated into three groups (road, road boundary, and linear object features) by comparing each point with neighboring points.

3. The traversability value of the cell corresponding to each point is adjusted depending on the confidence value of the point for each group. For example, if the point has a high confidence value for the road boundary feature, the cell should be adjusted to a non-traversable value.

There could be noise from the sensors, so the algorithm is likely to misclassify the noisy data. The risk of completely trusting the classification is obvious, so the confidence value is introduced. Three different methods, grid-based, height variance, and slope between points, are used to find the road boundaries. In the case where all three methods classify into the same group, the point has a high confidence value for the group. The information about the classification is converted onto the grid map and sent to the other component via JAUS messages to be used immediately or to be saved in the Sensor Knowledge Store for the future.

Sensor Knowledge Store

This section describes the Quadtree algorithm, which is one of the several possible methods that can be used to store an unknown amount of terrain data. The Quadtree algorithm is often used for visualizing large terrain. The Triangulated Irregular Network (TIN) method was recently developed but it has difficulty executing geographical operations, such as neighbor finding, searching, and updating. The purpose of using the Quadtree algorithm is to show an easy way to implement these operations.

The use of a Quadtree, where each node in the structure has four children, is used to an advantage for geographic data storage. If necessary, the geographic region can be divided into four sub-regions, to provide the resolution needed to effectively map or describe the environment. Large sections of contiguous terrain do not need to be divided, as increased geographical resolution is not needed in these regions. Detailed

areas can be divided into four regions repeatedly to obtain the needed resolution to accurately describe the environment. In this way, infinite resolution is theoretically attainable, while memory storage for the environment description is minimized. Chapter 4 describes how the Quadtree approach can be used to store the data from multiple sensors, such as LADARs, GPS, and vision sensors, to model a large driving area.

The primary reason for implementing this sensor knowledge store was to make path planning faster when the vehicle will be driving in an area that was already evaluated. For example, the sensors on the vehicle can estimate the environment around the vehicle for limited distances, i.e. 20 m ahead for terrain evaluation and 30 m for obstacle detection. This fact means the vehicle path planner can make decisions based only on this short range. Using this sensor knowledge, the planner can expand its range to longer distance unless there is a moving obstacle. Also this component can help generate intermediate waypoints between two waypoints since it already knows the environment around the waypoints.

A Quadtree is a tree data structure in which each internal node has up to four children, which represent NE, NW, SW, and SE. A Quadtree is most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions. In this study, the regions are square. The Quadtree is able to save any type of data, including double, integer, or char. There are two ways to implement the Quadtree algorithm, top-down and bottom-up. In this component the bottom-up method is used because the size of the area that the vehicle has to traverse is unknown. Using this method, the size of the area that is covered by the Quadtree can be built up. Figure 3-5 shows how the Quadtree is implemented. A larger quadrant is a node at a higher

hierarchical level on the Quadtree. Smaller quadrants appear at lower levels. The advantage of this structure is that regular decomposition provides for simple and efficient data storage, retrieval, and processing. As shown in Figure 3-5, if the (n-1)th level is 0.25 × 0.25 m resolution, then the nth level should have a 0.5 × 0.5 m resolution.

Vector-based Path Planner

A novel path planning method based on vectors will help the current planner, based on a grid map, have the opportunity to find a better path. The current planner is using different approaches depending on the vehicle's behavior.

First of all, a predefined speed profile is developed. The vehicle can know the course through which it has to pass by using the Road Network Definition File (RNDF), so the algorithm can define the best speed profile for the vehicle to follow. There are several elements such as road curvatures, the existence of obstacles, stop signs, and road roughness that can decide the speed at each segment.

While the vehicle is in roadway navigation behavior, the new planner can make the decision to find the optimal path more easily by assuming that waypoints are on the center of the traffic lane. Intuitively, if there is no obstacle and the road is smooth, an optimal path could be the path connecting each waypoint. With this optimal path, other candidate paths are decided by expanding the optimal path to the left and right laterally in constant step distances.

There are at least two difficult path planning problems in urban driving, the n-point turn and parking. Previous research in [76] based on A* for the DARPA Urban Challenge (DUC) has worked very well for these problems. To improve the algorithm, a new algorithm, Rapidly-exploring Random Trees (RRT), is introduced for the problem. The motion planning problem for non-holonomic platforms in the environment starts

from the first step that defines the 2D configuration space. Random nodes to be used for RRT are generated in the space. Then the algorithm expands the tree without collision and if it is successful in finding a solution following the goal criterion, the path will be smoother.

Bezier Curve Fitting

Bezier curves were widely publicized in 1962 by the French engineer Pierre Bezier, who used them to design automobile bodies. The curves were first developed in 1959 by Paul de Casteljau using de Csteljau's algorithm, which become a numerically stable method to evaluate Bezier curves.

A Bezier curve is a parametric curve that is sometimes used to generate a smooth path for autonomous vehicle path planning and related fields. In vector methods, Bezier curves are used to model smooth curves that can be scaled indefinitely. Vector-based paths are not bound by the limits of rasterized images and can be intuitively modified.

Bezier curves are also used in the time domain, particularly in animation and interface design, e.g., a Bezier curve can be used to specify the velocity over time of an object, such as an icon moving from A to B, rather than simply moving at a fixed number of pixels per step. The vector-based path planning uses this concept.

The curve can be implemented at different orders and requires (n + 1) control points for each order, where n is the order of the curve. Quadratic and cubic Bezier curves are most common and this dissertation also implements these two curves, since the higher the degree of curves, the more expensive it is to implement. A common adaptive method is recursive subdivision, in which a curve's control points are checked to see if the curve approximates a line segment to within a small tolerance. If not, the

curve is subdivided parametrically into two segments, *0 ≤ t ≤ 0.5 and 0.5 ≤ t ≤ 1*, and the same procedure is applied recursively to each half.

The formula of the generalized Bezier curve is

$$B(t) = \sum_{i=0}^{n} b_{i,n}(t)P_i = \sum_{i=0}^{n}\binom{n}{i}(1-t)^{n-i}t^i P_i, \quad (3\text{-}1)$$

where $b_{i,n}$ is known as a Bernstein basis polynomials of degree *n*.

The rational Bezier curve adds adjustable weights to provide closer approximations to arbitrary shapes. The numerator is a weighted Bernstein-form Bezier curve and the denominator is a weighted sum of Bernstein polynomials. Rational Bezier curves can be used to represent exactly segments of conic sections. Given *n + 1* control points P_i, the rational Bezier curve can be described by

$$B(t) = \frac{\sum_{i=0}^{n} b_{i,n}(t)P_i\omega_i}{\sum_{i=0}^{n} b_{i,n}(t)\omega_i}. \quad (3\text{-}2)$$

Both normal and rational curves are used for this dissertation.

Rapidly-exploring Random Tree (RRT)

Because the robotic path planning problem has received a considerable amount of attention, there are many proposed path planning methods. The problem of driving through a complex environment is embedded and essential in almost all car-like robotics applications. Since the urban area is usually a dynamic environment, moving obstacles have to be considered when planning.

The motion planning problem for non-holonomic platforms in the environment starts from the first approach that defines the 2D configuration space. This research introduces the Rapidly-exploring Random Tree (RRT) described in [77]. RRTs are constructed incrementally in a way that quickly reduces the expected distance of a randomly-chosen point to the tree. Figure 3-6 shows an RRT's result after 1000

iterations without non-holonomic constraints. The red-dot in the center of the figure represents the initial state of the tree. RRTs are particularly suited for path planning problems involving obstacles and several constraints. Usually an RRT alone is insufficient to solve a planning problem, so the algorithm can be considered as a part of planner that has many constraints and other methods.

There are several good properties of RRTs:

1. The expansion of an RRT is biased toward unexplored portions of the state space.
2. The distribution of vertices in an RRT approaches the sampling distribution, leading to consistent behavior.
3. An RRT is probabilistically complete under very general conditions.
4. The RRT algorithm is relatively simple.
5. An RRT always remains connected, even though the number of edges is minimal.
6. An RRT can be considered as a path planning module, which can be adapted and incorporated into a wide variety of planning systems.
7. Entire path planning algorithms can be constructed without requiring the ability to steer the system between two prescribed states

Figure 3-7 shows the nodes in the tree are expanded: *r* represents a generated random point, x_{near} is the closest point to the random point in the tree, and x_{new} represents the new input expanded from the x_{near}.

The approach to adapt this algorithm to urban driving for special behaviors, such as parking and n-point turns, are discussed in the next chapter.

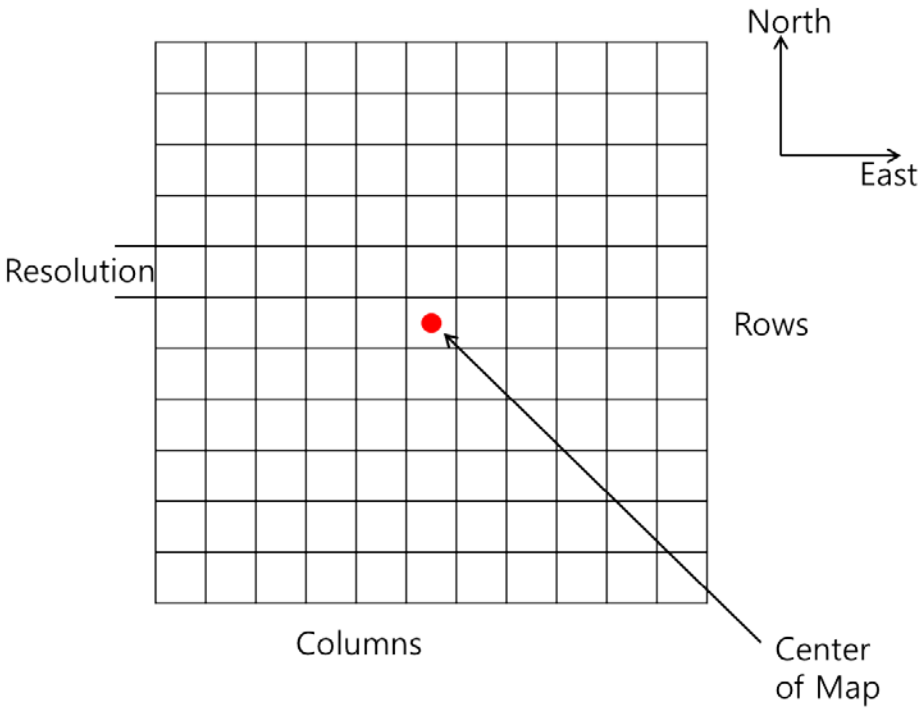


Figure 3-1. Traversability Grid Map

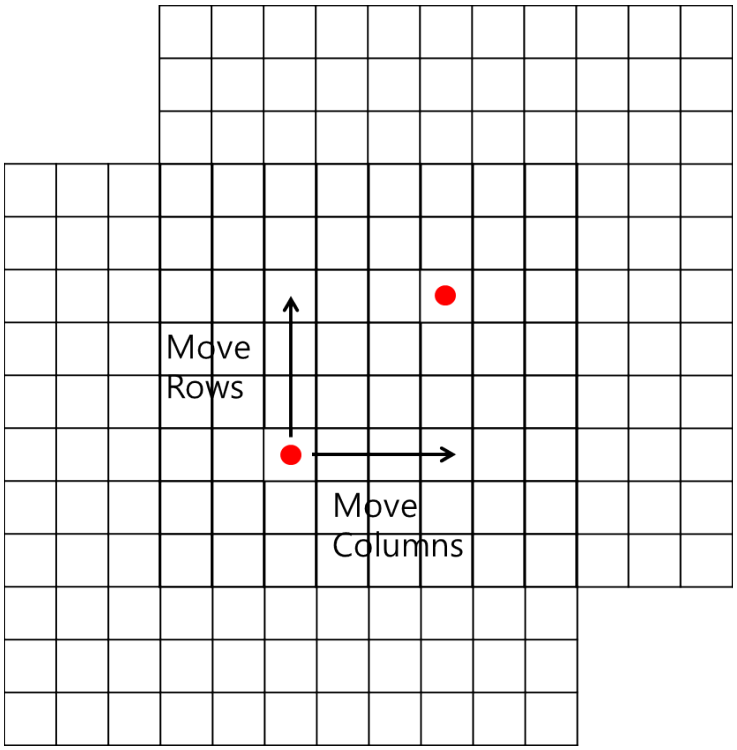


Figure 3-2. Grid Map movements

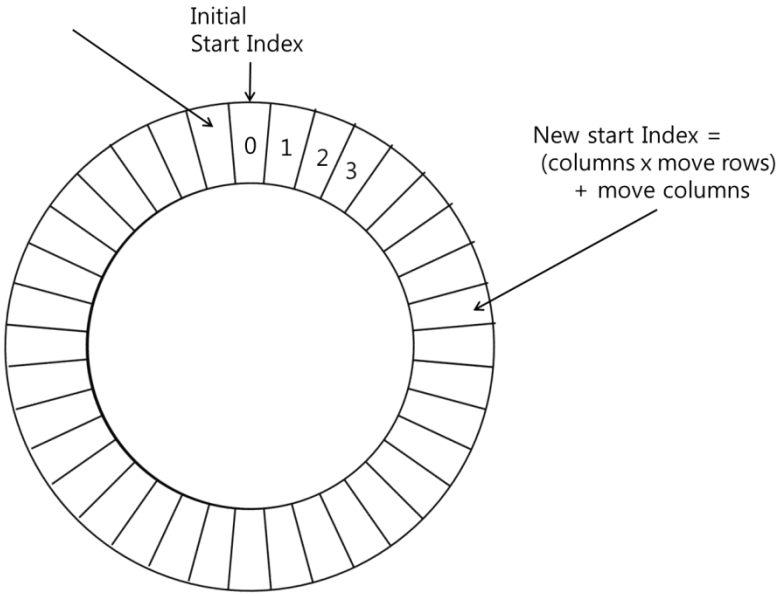


Figure 3-3. Circular Buffer

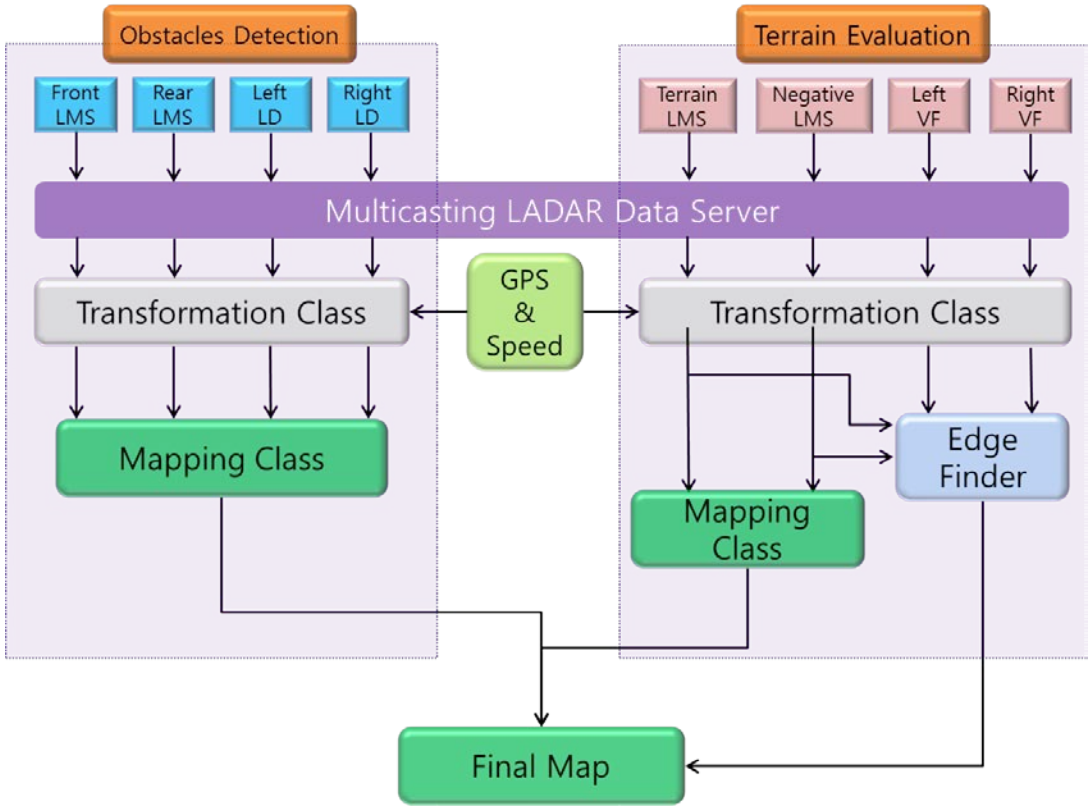


Figure 3-4. TSS diagram with Edge Finder

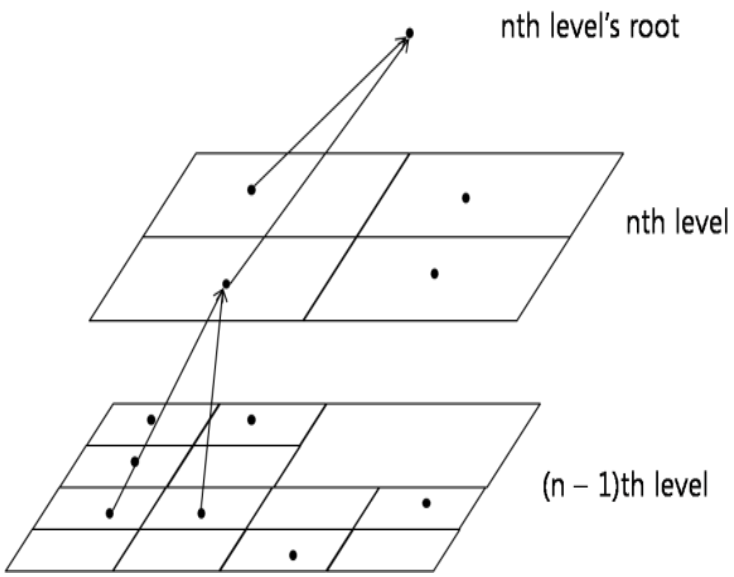


Figure 3-5. Hierarchical data structure Quadtree

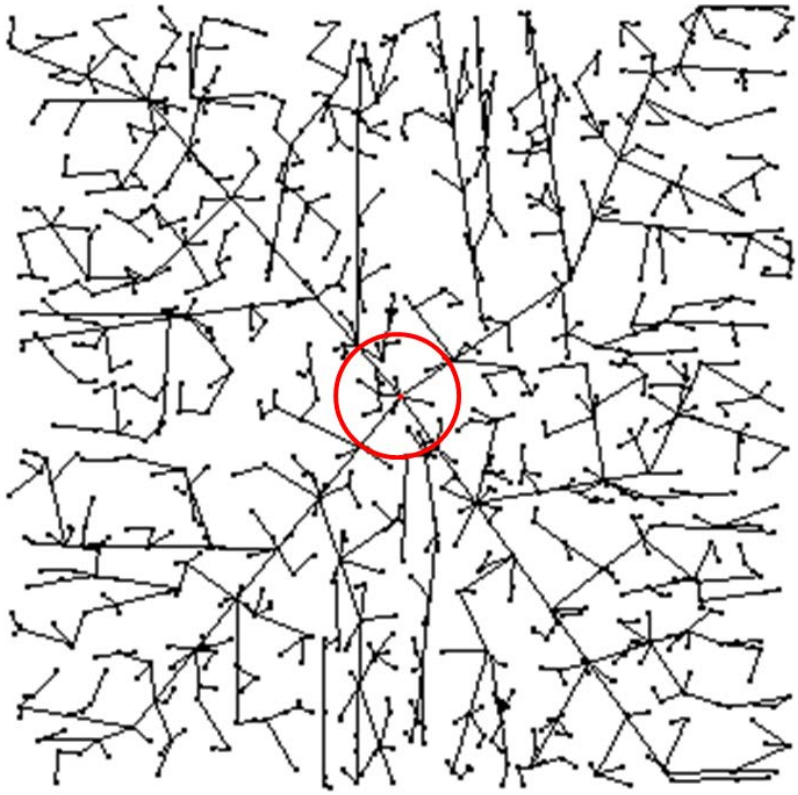


Figure 3-6. RRT graph after 1000 iterations

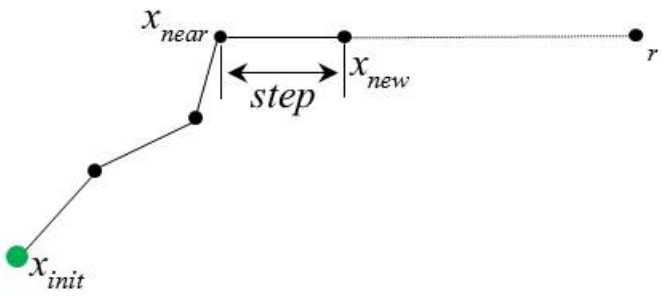


Figure 3-7. RRT node expansions

CHAPTER 4
IMPLEMENTATION DETAILS

This chapter introduces the technical details of the current implementation of the components for the research in this dissertation. The final objectives of the research are better sensing performance, effective and feasible storage of sensor data, and a novel path planning method using the sensor information for urban area driving for the autonomous vehicles. The present resources at CIMAR made this goal a reality in terms of software development and the ability to test the method on a reliable robotic test platform. First of all, the most recent robotic ground vehicle platform designed and built at CIMAR, the Urban NaviGator, is described and hardware equipment on the vehicle is briefly explained. Next, the software component to implement the components is discussed in detail.

Urban NaviGator

The Urban Navigator, shown in Figure 1-1, is a fully autonomous sport utility vehicle that was designed and developed at CIMAR for the 2007 DARPA Urban Challenge robotics vehicle competition and served as the robotic test-platform for the implementation and testing of the components explained in the previous chapter. The robot was built on a 2006 Toyota Highlander Hybrid (THH) chassis that was heavily modified to meet the requirements of the competition. The point of the hybrid vehicle is to autonomously run on its internal electric power train and/or its internal combustion engine.

Architecture

The system architecture of the Urban NaviGator is shown in Figure 4-1 and is composed of the four main elements: perception, planning, intelligence, and control.

The perception element contains all the sensor hardware and software components that provide data about the surrounding environment to the rest of the system. This includes three GPS units, eight LADAR sensors, four vision sensors, and the software components to perform such tasks as understanding the vehicle's position, detecting obstacles, estimating terrain, and traffic lane detection.

The planning element runs using two files: the Mission Data File (MDF) and the Route Network Definition File (RNDF). The MDF defines the locations the robot has to visit and the order in which the robot has to visit them. The RNDF defines the entire drivable environment as a set of GPS-based waypoints and check points to plan high-level and mid-level trajectories. The mid-level plans are provided to the low-level motion planning algorithm to calculate the goal point at which it aims its trajectory.

The intelligence element is responsible for collecting and analyzing information about the vehicle, the environment, the status of the mission, and the available operation behaviors to best choose the next course of action for the vehicle.

Lastly, the control element executes low-level motion planning, which generates the control inputs implemented by the vehicle to navigate through its environment avoiding static and moving obstacles.

Hardware

The vehicle is equipped with a comprehensive sensor package used for localization, terrain estimation, and obstacle detection. This package includes three SICK LMS 291 S05 LADAR sensors, three SICK LMS 291 S14 LADAR sensors, and two SICK LD-LRS 1000 long-range LADAR sensors for obstacle detection and terrain estimation. Also four Matrix Vision BlueFox USB color cameras are used to detect the traffic line and find the path. A Novatel GPS and two Garmin GPSs are combined with a

GE Aviation North Finding Module (NFM) for vehicle localization. The NFM estimates the vehicle's position and orientation in its global frame using Kalman filter estimation.

For all data processing, a computing package in the Urban NaviGator and a laptop, main computing unit, are used. The computers on the vehicle are powered by an ATX motherboard, AMD X2 46000 processors, and Ubuntu 8.04 and Windows XP operating systems. The main laptop has an i7 core processor and Windows 7 operating system. All the computers can transfer data through two gigabit Ethernet network switches.

Actuation of the Urban NaviGator is facilitated by several methods. For steering, an Animatics SmartMotor is attached to the steering column and receives position commands that are associated with the steering commands coming from the motion planning algorithm. Throttle and brake automation is enabled through the use of the existing drive-by-wire system already implemented on the vehicle. A custom controller was designed and incorporated to pass throttle and brake commands to the vehicle when it is in an fully autonomous state.

Road Boundary Detection

One of the reasons that the autonomous navigation is still a considerable challenge is the difficulty of describing the environment to the robot in a way that captures the variability of natural environments. For example, there are many different terrain conditions, such as short or tall grass, forested areas, and trees, that are naturally described as having a 3D texture rather than a smooth surface.

In contrast to off-road driving, in urban driving there are many more ground conditions in which vehicles can or cannot traverse. For example, the curb, which is the boundary of the street, is a non-drivable condition, but for off-road driving the curb is a

drivable area because it has little or no height. Thus urban driving needs more precise evaluations than off-road driving.

In this section, three ways to detect road boundaries of urban areas are introduced: the peak detector method, the slope variance method, and the grid-based method. For these methods, four LADAR sensors, Terrain, Negative, driver-side Vertical Fan (VF), and passenger-side VF, are used shown as Figure 4-2.

Peak Detector

Peak detection, for one-dimensional or multidimensional signals, is often encountered in signal processing. Finding peaks in experimental data is a very common computing activity and many techniques have been established. In this dissertation, a 1D signal is used along with a method that is based on a Savitzky-Golay smoothing filter [78]. This peak-finding technique uses the zero crossing derivatives of the smoothing, locally fit, Savitzy-Golay polynomials. This is a very fast peak detector because the Savitzy-Golay smoothing algorithm can be slightly altered to directly report the first derivatives.

The input signal for this method is the variance of the z-height of the point clouds. The variance of the points is computed in a group of five neighbor points that is composed of itself, the previous two points, and the next two points. In this dissertation, the group is named a *unit*. The important thing is that in the unit there could be invalid data points that have longer distances than the maximum distance of the sensors, so the algorithm needs to filter these points out. One unit for the other two methods is also defined like this.

First, the algorithm needs to transfer the LADAR beam into the proper coordinate frame, depending on the type of the sensor. The input data should be in an x-y

coordinate system, where y is the z-height of the data points. For the Terrain and Negative sensors, the x values of the data points in the vehicle frame can be easily used, since the sensors are fixed toward the front of the vehicle. However, the VFs sweep with smart motors, so the x values of the input data needs to be transferred from the x value in the vehicle coordinate system as follows using the smart motor position shown in Figure 4-3.

$$x_{vf} = x_{veh} * \cos(-\theta) - y_{veh} * \sin(-\theta), \quad (4\text{-}1)$$

where θ is the VF's angle. Once the transformation is complete, the input data also can be used for the other two methods.

The second step of this method is to smooth the first derivative of the y values, which are the z heights of the data points. The first derivative that represents the average slope between three adjacent points can be expressed simply:

$$y' = \frac{y_{j+1} - y_{j-1}}{2\Delta x}. \quad (4\text{-}2)$$

At this point, extra grouping is necessary to find local maxima. Each group has a specific number of points that is 15 in this dissertation, which is the group width, and the number of points should be odd so the mid-point is the center of the group. The fast smoothing is operated in each group as follows:

$$S_i = \frac{\sum_{j=i-(m-1)}^{i-1}(j+m-i)*y_j + m*y_i + \sum_{j=i+1}^{i+(m-1)}(m-i-j)*y_j}{2*\sum_{j=1}^{m-1} j+m}, \quad (4\text{-}3)$$

where m is the half width of the group.

The last step is least-square curve-fitting for the groups of the points whose smoothed first derivative is higher than the slope threshold value and the computed y

value is higher than the amplitude threshold value. The curve-fitting is operated in 2nd-order polynomial fitting:

$$y = a + bx + cx^2. \quad (4\text{-}4)$$

Once the coefficients a, b, and c are computed, the peak position, height, and width can be found using the quadratic coefficients and equation for a Gaussian peak:

$$h = e^{a-c*(b/(2*c))\text{^}2}, \quad (4\text{-}5)$$

$$x = -b / (2 * c), \quad (4\text{-}6)$$

$$w = 2.35703 / (sqrt(2) * sqrt(c)), \quad (4\text{-}7)$$

where *h, x*, and *w* represent peak height, position, and width, respectively and 2.35703 is from the formula of the Gaussian peak.

The height of the peak is necessary to distinguish the road boundaries from vertical obstacles, such as cars and walls. The road boundaries have relatively lower peak heights compared to vertical obstacles.

Determining positions of the peaks is necessary simply to let the vehicle know where the edges are. In addition, ascertaining the width of the peaks supplies the range of the edge that can be considered as edge. The reason the width is necessary is for formulating a confidence value of the detected road boundary. The confidence value rises as more sensors detect the edges at the same position using multiple methods. To match a detected edge by this method with an edge using other methods, some offset of the edge is necessary, since it rarely happens that the two points are identical. For this reason, the width of the peak is needed to add to the confidence value of the peaks.

These procedures using sample data are shown in Figure 4-4. The top three graphs are z-height road profile, standard deviation of five points of z-height, and

smoothed z-height using equation (4-3). The fourth figure from the top shows the fitted curve using points in the blue circle using equation (4-4). The bottom figure shows the final result.

Slope Variance Method

Humans can detect most road boundaries by change of height, color, or materials. The LADAR sensors that are used for this dissertation, however can only detect the change of the height from the road profile and other changes that can be detected by vision sensors, so the basic concept, to detect the road boundaries, starts with determining changes of the height. One thing that should be considered is that there can be misclassifications when trying to detect the change in *absolute* height, since there will be uphill and downhill on the road that the vehicle has to traverse. For this reason, detecting dramatic change of the slope between the point clouds is very helpful to detect these variations in elevation.

This method, using the variance of the slope shown in Figure 4-5, is very simple, but works very well. In the figure, the red dots represent the points hitting the curb area and the black arrows are orientations of the slope between two points. The slope (S_i) of each point is computed relative to the previous point. All the points are in the coordinate frame that has been obtained from the previous section for Peak Detection and the slope is calculated as:

$$S_i = arctan(\frac{y_i - y_{i-1}}{x_i - x_{i-1}}). \quad (4\text{-}8)$$

Filtering the invalid points out is an important step. It is highly likely that using a threshold value for the slope will result in misclassification. For example, if the road has a gradually declined (or inclined) profile, the vehicle can drive on the road. However, if

the road slope is higher than the threshold, it could be classified as a non-traversable edge. The variance of the slope in unit, defined in the previous section, is used to prevent this. This variance indicates how much the difference of the slope of the points in the unit is. The higher variance usually can be derived from the rough terrain or the unit that contains the LADAR beam hitting road boundaries such as curbs. The variance of the unit is expressed as:

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^{n} (S_i - \mu)^2, \quad (4\text{-}9)$$

where μ is the mean of the slopes in the unit and *n* is the number of slopes that are one less than the number of points in the unit. What if all the points in the unit are hitting objects, not road class, such as walls or curbs? The variance must be less than the threshold value and the point could be classified incorrectly. This can be prevented simply by introducing another threshold value for the slope of the point that should be considered road group.

Grid-based Method

The road boundaries were detected using the previous two methods, but this may not be enough and there could be noisy values. To minimize noisy values and increase the confidence value, a new method, the Grid-based method, is introduced. This Grid-based method is different from the classic fixed sized grid map method. The grid size can be changed by the number of the valid LADAR beams and the total width that each sensor covers. This method can also use the point clouds computed at previous sections.

The experiment shows the average of the total width of the LADAR beams is enough to determine the grid size of each cell for the sensors on the roof since the

lengths of the beam of the sensors in the x-direction are equal when there are no external factors such as yaw, pitch, or roll angle. However, the VFs need the variant cell size, depending on the length of the beams. The longer the beam length is, the wider the size necessary and vice versa. Each grid size is computed with a simple formula,

$$w_i = k * (r_i * \theta), \quad (4\text{-}10)$$

where r_i is the length of beams, θ is the angle between two points, and k is the width-tuning constant that make the sum of the sizes of grids equal to the total width of the LADAR beams.

Once the grid size is decided, the each point falls into a corresponding cell by its position and the detection process is starting.

This method uses the number, the maximum height difference, and the slope change rate of the points in each cell. The points in cells that are in road features are usually distributed laterally, so the cell has two or three data points at most with low height difference and slope change rate. The points in cells that are in road boundaries (or linear obstacles) are distributed vertically, however, so the cell has more than three points with a higher height and slope change rate than the road feature cells have. Based on these properties, cells that have the beams hitting the road boundaries can be distinguished from the drivable areas. Figure 4-6 shows examples for this method resulted from the sensors on the roof (upper) and VF (lower). As shown in the figure, the width of each cell is constant for all beams for the sensors on the roof, but the width of each cell varies depending on the beam length of the VF sensors.

After extracting such road boundaries as curbs, vertical objects, or walls using these three methods, a confidence value for the edge is assigned to each point. Then

the three confidence values are applied to the cell corresponding to the point on the grid map and the traversability values of the cells will be modified depending on the confidence value.

Sensor Knowledge Store

This section describes how the Quadtree approach was used to store the data from the sensors to create an accurate model of the operating environment. The algorithm is flexible and allows for the storage of virtually any data type. Examples include the mapping of traffic lines, obstacles, and terrain elevation. In the next chapter experimental results are provided that show the operation of the algorithm in realistic driving environments. The Quadtree component is composed of the Quadtree Generator and the Quadtree Extractor as shown in Figure 4-7.

Storage of Sensors Information

For autonomous driving it is necessary to store and integrate several sensors' information, such as that from the obstacle detection sensor, the terrain estimation sensor, and the traffic line detection sensor. This information is placed into the appropriate node of the Quadtree corresponding to the vehicle position as obtained from GPS and inertial sensors. All the sensors' information are reported from the other components with a different update rate than the Quadtree generating rate, so it is likely that this component could miss reported data. To prevent this, the component has a queue, so incoming data are queued in order.

The Quadtree is able to store all the information from the various sensors. If the point to be stored is in the range the Quadtree covers, the procedure for this approach is two steps, as follows:

1. A leaf node corresponding to the new information has to be found. To find this node, the new point needs to traverse h (height of the tree) nodes at most until reaching the leaf node from the root node of the tree. In order to traverse to the leaf node, the position NE, NW, SW, or SE of a child of the current node to which the point has to move should be decided by comparing the center of the current node and the position of the data point to be saved and the center of the child has to be computed. The center and position can be saved to omit this computation, but since one of the requirements for this approach is to minimize storage usage, they are not saved and the positions of all the nodes can be computed from the position of the root node of the tree. For this step, all the positions are represented by a Universal Transverse Mercator (UTM) coordinate system value transformed from geographic coordinates that are composed of latitudinal and longitudinal values.

2. The next step is to store the new data in the node or update the data in the node by comparing new data. The data are composed of the roughness of terrain, static obstacle existence, traffic line existence, and way point existence in specific bits of a char data type shown in Table 4-1. In addition, the measured mean height in the node that is usually double data type is saved as converted char type data. There are two reasons to save these data into char type data. One is to reduce the necessary storage memory. The other is to increase the speed of sending the message extracted from the tree to the path planner or some other component that needs the plane model information. This will be described later.

In case the new data point is out of the Quadtree range, the Quadtree needs to grow larger and taller by finding a new center position of the potential new root. This is described in the next section.

If the vehicle will be running the same loop repeatedly, the data will be stored during the first run and the data in the leaf node will be updated from the second run. The update operations are composed of traffic line adjusting, obstacle re-detection, and terrain re-evaluation operations. This can act as an extra sensor by providing the information that is already saved during the first run when any of the sensors does not work properly due to a hardware failure, communication problems, weather problems, or dim light and only GPS sensor can provide the vehicle’s position. Unless there are moving obstacles or new obstacles, it can work very well.

The formula for the time complexity of search and insert (or update) of this approach is $O(lgn)$ where *n* is the number of nodes in the tree, since the height of the tree can be computed as lgn.

Bottom-up Quadtree

The previous section described how to insert a node and update node data when the new data are in the Quadtree range. This section, one of the core parts of this component, describes how a new tree is built when new data are out of the current max range.

Since the vehicle does not know the size of the area that it has to traverse, the algorithm does not know how large the Quadtree should be before the vehicle finishes driving. Although the vehicle will be provided a set of waypoints that it has to traverse, it cannot anticipate how big an area the sensors will cover. So if the size of the area where the vehicle has traversed is greater than the maximum range of the current Quadtree, the current root becomes one of the children (or descendants) of the root node that is newly generated. If the old root became a descendant that is not a child, the algorithm has to make connecting nodes between it and the new root.

In this approach, a leaf node's area size is 0.25 m resolution and the maximum size of the tree with current height *h* will be $0.25 * 2^{(h-2)}$. The level of the leaf node is always 0. In addition, one can use a smaller resolution, such as changing from 0.25 to 0.125 m, although the required memory allocation will be larger.

Figure 4.8 shows how the bottom-up Quadtree is built. In the far left figure there is only one node, which consists of a root and leaf node. In this case, the root node (also leaf node) has a 0.25 × 0.25 m resolution. Then, as the Quadtree gets bigger, (center of Figure 4-8), a new data that has to be saved is out of the current range, so the root

node becomes one of the children of a new root. The fact that the old root becomes the child of new root means the difference between the level of the old root and new root is one. The procedure from left to right in the figure represents the old root becoming one of the children of the new root. The same procedure is applied to growing up from the middle of Figure 4-8 to the right figure, but the level of the new root is two levels higher than old root. In this case, the old root becomes one of the descendants of the new root, so they need to be connected by generating new intermediate internal nodes between them. Figure 4-9 shows the algorithm of bottom-up Quadtree building.

The time complexity of building a new root and connecting a new root to an old root is $O(lgn)$ for the same reason as that in the previous section.

Extraction of Quadtree

The Quadtree component is composed of the Quadtree Generator and the Quadtree Extractor (Figure 4-7). The Quadtree Generator is the function that constructs the Quadtree using the method described in the previous sections.

The other function is the Quadtree Extractor. The concept of the Quadtree Extractor is quite simple. This function is necessary to send terrain information of the requested area to the other components. The other components that need the ground data make and send a request message with the start point and goal point of the necessary area. Then leaf nodes (or the nodes at the desired level) of the tree inside the rectangle defined by the start point and the goal point are extracted and the information is stored in an array with the corresponding resolution.

Before starting extraction, if the length or width of the area to be extracted is shorter than minimum values, they should be adjusted to minimum values. With

adjusted points, the number of rows (*r*) and the number of columns (*c*) of the grid map to be reported are decided.

The first step of the extraction is to find the lowest common ancestor (LCA) node of the start point and goal point to lower the searching start point. The LCA node can be found by comparing the position of the start and goal points relative to a node. For example, if the positions of the children of a node for the goal and start points are same, there is a lower common node but if the positions of the children of a node are different (e.g. NW for start and SW for end point), the node is the LCA. Once the LCA node is found, the extraction process starts from that node. Figure 4-10 shows the extraction of the necessary area of the built Quadtree. In the figure, the start point and the end point queried from other components are shown in a grid map and corresponding nodes in the tree are shown. The LCA node of two points is found in the tree, and then the searching process runs in the tree, shown by the purple line.

The next step is to search for a corresponding leaf node (or node with the desired resolution) to each cell from the LCA and bring the height and environment data in the node to the cell of the grid map. This search step runs from the start point to the end point of the grid map. There could be a case where the corresponding node to a cell does not exist. In this case, the cell simply remains an unknown data cell.

The time complexity of the this process is $r * c * O(lgn)$, where *r, c*, and *n* represent the number of rows and columns of the grid map and the number of nodes, respectively.

As shown in Figure 4-7, the path planner (or another component) queries the Quadtree Extractor with a message composed of the requested ID, the start point, the

goal point, and the requested resolution. The Quadtree Extractor then searches the area and picks the information and stores the char type array. The array is then converted into the message with the requested ID and adjusted start and goal points and the message is sent to the other components.

Figure 4-11 shows an example of the extraction. The upper figure is a display of the current Quadtree. In this case, the difference of the *y* coordinate between start point and goal point is smaller than the threshold value of 20 m and the distance between the *x* coordinates is 50 m. The height of the map is set as 20 m. The nodes in the green box are to be extracted.

The lower figure is the result of the extraction. The size of this map is 80 × 200 cells with 0.25 m resolution. The cells colored yellow represent the leaf nodes that have the data of the terrain, other cells are null status.

The data structure of each cell of the extracted grid map is shown in Table 4-1. The first two bits are reserved for future use. Obstacle existence is located at the 5^{th} bit and is updated every loop if the existing obstacle is a moving obstacle. Messages among the components to report a set of data from the LADAR sensor components to the storage component and from the storage to the other components are shown in Table 4-2.

Save and Reload

Another feature, saving the Quadtree when the program is terminated and then reloading the previously saved tree when next starting the program, is useful when the vehicle drives in the area where it has been before. In this case, using waypoints from the Roadway Network Definition File (RNDF), the planner can create a path and set the vehicle speed based on the static obstacles, traffic lines, and terrain roughness that is

stored in the Quadtree. Then when driving around the course, the vehicle will acquire local environment data from the sensors and if there is any new obstacle or environment information, such as a moving obstacle, the data in the Quadtree will be updated.

Path Planner

The remaining problem of the approach in this dissertation is path planning to drive the UGV without obstacle collision. Many researchers have proposed a variety of methods for this problem such as A*, D*, graph-based, and so on. All the methods have advantages and disadvantages. This section describes how to generate candidate paths and choose an optimal path from them for the UGV for urban area driving, such as roadway navigation, parking, and n-point turns, using sensed information mentioned in previous sections.

Occupied Grid

One of the critical functionalities of path planning is to check for potential obstacle collision. This can be done faster by introducing the grid map method. This method detects the cells occupied by obstacles and checks if the planned paths pass through the cells. If so, the path will be ignored or much more cost will be added to the path.

The traditional way for CIMAR to avoid obstacle collision is, shown in Figure 4-12:

1. The vehicle is assumed to be one cell of grid map. Yaw angle is not considered. In Figure 4-12, the vehicle is the black cell at the center of the grid map.
2. The cells occupied by obstacles are dilated with constant diameters since the vehicle is a rectangular object not a point. Yellow cells in Figure 4-12 represent the obstacles that are expanded with constant diameter.

This approach has some problems, however:

1. The vehicle is not a circular object, which means the length from its center of gravity to the border is different depending on the direction. For example, length to the front bumper from the center is longer than to the rear bumper from the center.

2. Many experiments have shown that, although the vehicle can pass between two obstacles, dilation with constant diameter could interrupt this run. Figure 4-13 shows this problem. In Figure 4-13 (a), the car that is the green rectangle can pass between the obstacles that are orange rectangles. However, in Figure 4-13 (b), the expanded obstacles based on the constant diameters make the car unable to pass between them.

To resolve these problems, a novel way to check for potential obstacle collision is introduced. This method assumes the vehicle is a rectangle, not a point, and uses the grid map to quickly check it. It expects position and heading angle of the vehicle on each node of the planned paths, finds a set of cells that are occupied by the rectangular vehicle shown in Figure 4-14, and checks if the cells have obstacles or not. In this figure, there are five possible paths. A series of blue boxes represent the estimated vehicle's position on each node of a path. The red paths and blue paths represent the ones with and without potential obstacle collision, respectively. This method also can be applied when the planner estimates the cost of the paths using terrain roughness. The path planner that is developed in this dissertation uses this novel method to check potential obstacle collision.

Vector-based Path Planning for Roadway Navigation

Many proposed methods for path planning work very well for off- and on-road driving. However, the path planner in this dissertation is focusing on driving in urban areas. The paths can be generated more easily for roadway navigation in urban areas by introducing two assumptions:

1. All the waypoints that the vehicle has to pass are on the center point of a traffic lane.

2. The center point of traffic lanes is smooth enough for the vehicle to traverse.

With these assumptions, the center of candidate paths can be generated by connecting the current waypoints that are given by the RNDF providing component and, if there is no obstacle on the path, this is considered an optimal path at the last moment. A set of candidate paths can be expanded from this center path laterally with constant distance and the number of paths and the distance are decided depending on the width of the traffic lane in which the vehicle is located. Usually, the number of candidate paths on left and right sides of center are even and the closer the paths, the lower the path cost. Figure 4-15 shows this path planner.

The path planner for roadway navigation is composed of short-range paths and long-range paths. The short-range paths are established for a smooth transition from the current position to a certain position on a collision-free path. The length of the paths is decided by the current vehicle speed. Intuitively, the vehicle needs to rotate the steering wheel more slowly at high speeds for stable driving, so the length should be longer for smoother paths. The paths are generated using the cubic Bezier curve equation. The parametric form of the curve is

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 + t^3 p_3, \quad (4\text{-}11)$$

where $t \in [0, 1]$ and *P0-P4* are control points. The selection of control points is shown in Figure 4-16. *P0* is the vehicle's position, P_1 is the point *a* meter ahead, P_2 is the point *b* meter toward the vehicle direction, and P_3 is the lateral point of the point in look-ahead distance from the vehicle. P_0 and P_1 are invariant and P_2 and P_3 are variant, depending on the index of lateral path. The generated short paths are connected to long-range paths.

Long-range paths are generated by simply connecting end points of the short-range paths and the lateral points of the current waypoints. Usually these paths are used to check if the vehicle will collide with any obstacle when the vehicle follows a certain path.

In addition, each section that is a line connecting two adjacent waypoints has a predefined recommend speed to help path planning. This speed profile is published by considering the existence of a stop sign, terrain roughness, and static obstacles reported from the Sensor Knowledge Store, traffic line, and road curvatures. Figure 4-17 shows the predefined speed profile where the red circles represent stop signs. The thicker blue lines represent higher speed and the lighter blue lines represent lower speed.

With this new path planning algorithm, the vehicle can generate the candidate paths and find an optimal path without much memory usage during the roadway navigation behavior that happens most frequently as the vehicle is driving.

Vector-based Path Planning for Parking Behavior

One of the most challenging behaviors in urban driving is the parking behavior. In this dissertation, two types of parking lots are considered: a typical parking and a parallel parking lot. To approach this problem, the Rapidly-exploring Random Tree (RRT) algorithm, which is mentioned in Chapter 3, is used to find routes from the vehicle's position to the goal parking lot. However, the vehicle that is the test platform has a *nonholonomic* character, so several constraints are considered, such as steering angle limit, angular speed, vehicle shape, and so on.

The parking process for the two types of parking lots is shown in Figure 4-18 and there are three sub-behaviors: TO_PARK_LOT, FROM_PARK_LOT, and TO_EXIT.

RRT approach

Rapidly-exploring Random Trees (RRT) is an incremental method to quickly explore an entire configuration space. The first challenge for this algorithm is to set the configuration space. In this dissertation, the configuration space is set by the function in terms of the vehicle's orientation (V_h), the desired goal's orientation (G_h), and the distance between the vehicle and the goal (d), where the difference between V_h and G_h is called the *heading variation*, expressed as:

$$C_s = f(V_h, G_h, d). \tag{4-12}$$

Using equation (4-12), when the vehicle arrives at the position near to where the parking algorithm has to start, the configuration space is defined. During the process of setting the space, if the heading variation or distance is less than the stated threshold, the minimum variation or minimum distance is assigned to the function. Furthermore, if the algorithm cannot find the optimal path to get to the goal, the variation and distance increase by a defined step size to make the space bigger. An example of the configuration space is shown in Figure 4-19, where the green rectangle, the light blue rectangle, and the space filled with pink points represent the vehicle, parking lot, and configuration space, respectively. Random nodes will be generated in this space and the algorithm operates with the nodes.

The most crucial part of the algorithm is tree extension. The proposed method is bi-directional RRT, so there are two tree extensions: forward tree and backward tree extension. The only different thing between these two trees is x_{init} and x_{goal}. The x_{init} for the forward tree and backward tree are the vehicle position and parking lot position, respectively. Alternatively, the x_{goal} for the forward tree and backward tree are the

parking lot position and vehicle position, respectively. Except for this difference, the trees are extended as follows.

The extension part can be divided into five parts as: generating random node, x_{rand}, in $\mathcal{C}_s$; finding the nearest node x_{near} using x_{rand} from *k*d-tree; computing new input, x_{new}; checking for collision; and adding x_{new} to the tree graph *G(V, E),* where *V* and *E* are nodes and edges, respectively, if it is in collision-free space.

The closest node of the nodes in *G* to the x_{rand} can be found by the *k*d-tree, where in this dissertation *k* equals two, since the path planner assumes that the vehicle is driving on a plane. The root of the *k*d-tree is the initial point of the vehicle. For simplicity of the proposed method, the closest one has a minimum Euclidean distance.

If the x_{near} is found successfully, it is necessary to find a new input extended from x_{near}. To do this, a random steering angle, δ, is generated in the range of $-\delta_{\max}$ to $\delta_{\max}$ with a normal distribution. Using a vehicle dynamic equation derived from [79], δ and the step distance, which is three meters in this research, will yield a new status of the vehicle, x_{new}. The next step is to check the collision-free status of x_{new} using the proposed occupied grid method. The final step is undertaken if and only if the x_{new} is obstacle-free, $x_{new} \subset \mathcal{C}_{s,free}$. If it is obstacle-free, the x_{new} is added to *G* and the *k*d-tree. The new edge connecting x_{new} and x_{near} is also added to *G*.

The above steps are terminated when the elapsed time, t_{loop}, is longer than the minimum searching time, which is 0.1 sec in this approach, and the generated candidate paths are more than a minimum number, which is ten, or the elapsed time, t_{loop}, is longer than time limit, t_{limit} which is 3 sec in this research. Figure 4-20 shows examples of the tree extensions (upper: forward, lower: backward).

The candidate paths literally mean the available paths for the vehicle to get to the parking lot. There are four criteria to check if the new node is assumed to get to the goal position: *GOAL_REACH, CIRCLE_REACH, CONNECT_TREE*, and *INIT_POS_REACH*.

Intuitively, if the distance between the new node of the forward tree and the goal position is less than the threshold value and the difference between the orientation of the new node of the forward tree and the desired orientation is less than the threshold value, it can be assumed that the node arrives at the goal position and it is correct to add the goal node, not a new node, into the graph (*GOAL_REACH*).

However, from experience, the node that is assumed to be the goal may not be found in the allowed max search time. To make the rule more relaxed, the minimum radius circle of the vehicle is introduced whose radius is

$$\rho_{min} = \frac{L}{\tan(\delta_{max})}, \tag{4-13}$$

where L is the wheel base distance and δ_{max} is the maximum front wheel angle.

Figure 4-21 shows how to check if the new node in the min circle is useful or not. The left and right radius circles have the trajectory in clockwise and counterclockwise directions, respectively. The procedure to check this condition is written as:

1. Decide the side of the circle (left or right).
2. Compute the orientation, α1, of an edge between x_{new} and x_{near}.
3. Find the intersection point of the edge and the circle and compute tangential angle, α2, in the proper direction of the circle's path at the point.
4. Compute the angular difference between α1 and α2, then decide if the node and edge are useful or not.

Once the new node and edge are deemed useful (*CIRCLE_REACH*), a set of the path from the tangent point to the goal point following the circle is added to the candidate path.

Another condition for the goal reach is to check if the forward tree and backward tree are connected properly in Figure 4-22 (*CONNECT_TREE)*. The conditions to connect two trees are:

1. Distance between a node of the forward tree and a node of the backward tree will be shorter than the threshold distance.

2. If point B of the forward tree and point C of the backward tree are connected and the B is expanded from point A, the algorithm can compute the difference of orientation of $\overrightarrow{AB}$ and $\overrightarrow{BC}$. To connect two trees, this orientation difference will accommodate the vehicle's steering limit.

If conditions 1 and 2 are satisfied, the route from the vehicle position to point *B* of the forward tree (route 1) and the route from point *C* to the goal position of the backward (route 2) are connected and the connected route is registered as the candidate path. To acquire route 1 and route 2, the Depth-first search (DFS) in Figure 4-23 algorithm is used.

The last condition is *INIT_POS_REACH* where the new node expanded from backward tree reaches to the vehicle's initial position. This condition is similar to condition *GOAL_REACH*, but the initial point of the forward tree becomes the goal point for the backward tree and the goal point of the forward tree becomes the initial point for the backward tree.

Typical parking behavior

Now the planner has a set of candidate paths that are blue lines shown in Figure 4-24. In the figure, several paths do not touch the goal point since they reach to one of the minimum radius circles.

The next step with the paths is to find an optimal path for them. The heuristically computed cost is given for each path. The factors that are considered to compute the cost are length of the path, the sum of the vehicle's heading change, and type of goal to be reached. So the cost (C_i) of the each path, i, can be expressed as

$$C_i = k_d * D_i + k_h * H_i + R_i \ , \quad (4\text{-}14)$$

where k_d and k_h represent heuristic gains of length and heading change, respectively and R_i is the cost depending on the goal reach type. In this dissertation, the path that reaches to minimum circle is set as higher cost than other types since it is harder for the vehicle to follow the circle trajectory exactly unless it is a neutral steering vehicle. With these costs, the optimal path is selected.

The optimal path is the shortest path with the least orientation change, but it still could be worse than human driving or the path generated by other methods, since it is computed by random points. The optimal path is the white line in the Figure 4-24. For this reason and the smooth behavior of the vehicle, the simple algorithm in Figure 4-26 is used to smooth the path and yellow line in Figure 4-25 represents the final smooth optimal path.

Simply stated, the algorithm shows how to find the smallest number of the straight line segments. Line 7 of the algorithm determines if the straight segment is usable or not. To decide this, the straight line has to be collision free. Also, the difference between the previous line segment and the new line segment has to be drivable. For example, the left side of Figure 4-27 shows a very sharp corner between two segments and it is very difficult for the vehicle to follow the path. If the new segment is like that shown on

the right side of Figure 4-27, there is one more necessary step to make the corner smooth. The arc of a circle is inserted between the segments.

For parking using RRT, there are two options to generate a path to the goal position. One is a static path planning algorithm that generates an optimal path when the vehicle enters the parking area and just follows the path, and the other one is planning a new path every loop of the algorithm. Each has advantages and disadvantages.

The advantage of the static path planning algorithm is that it makes it easy and stable for the vehicle to follow the generated path, but the disadvantage is that the reaction time to avoid new or moving obstacles is longer than the replanning algorithm. The advantage and the disadvantage of the second way are exact opposite of the first way.

After the parking behavior is completed, the vehicle could try to back out of the parking space and exit from the parking lot. For these movements, the same procedures as for the parking behavior are executed.

In this dissertation, the replanning method is selected, but moving one method to the other method is actually very simple.

Parallel parking behavior

The other type of parking is parallel parking. When the vehicle is trying to park in a parallel parking spot, it needs to arrive to the virtual parking spot first using the RRT algorithm mentioned in previous sections. Figure 4-28 shows the parallel parking scenario. The yellow rectangle represents the virtual parking spot and it is laterally

located *L* (constant offset depending on the vehicle size) meters away from the real parking spot.

Once the vehicle stops at the virtual parking spot, it needs to estimate the parking spot size using vision sensors or LADAR sensors. This process extracts a rectangle as a real parking spot and the vehicle will decide if the parking spot is valid or not.

If the vehicle decides the spot is valid, it starts the parking process as follows:

1. Go forward until position ① in Figure 4-29. Position ① is decided by the parking spot's size, the vehicle speed, and the steering constraints. This position could vary depending on the vehicle. In this dissertation, position ① is decided depending on the minimum turning radius of the vehicle. In the US and other countries where the driver sits on left side of the car, parallel parking lots are on the right side of the road, so position ① is the moment when the right min radius circle of the vehicle touches at a point tangentially the left min radius circle of the expected car position when it is completely parked.
2. Go backward by following the two arcs in red in Figure 4-29. If the vehicle can follow the arcs completely, it can park easily without the steps after this.
3. Check for potential collision at the rear and side of the vehicle.
4. If there is any potential collision, then it should stop and control the steering and accelerator to get to the goal position.

During the process from 1 to 4, the vehicle keeps checking if it is completely in the parking spot. A simple feed forward controller is used to control the vehicle's steering, i.e.,

$$\delta = atan(k_h * e_h), \tag{4-15}$$

where k_h is the gain for the heading error and e_h is the heading error. This control equation is using only the heading error between the vehicle's orientation and the parking spot's orientation, since if any part of the vehicle is in on the parking spot the lateral error is small and can be ignored.

When the vehicle is parked completely, it could try to come out from the parking spot and this operation can be done using steps 3 and 4 of the parallel parking process. When the algorithm decides that the vehicle is out of the spot, it starts planning using the RRT procedure to go to the exit of the parking area.

Path Planning for N-point Turn Behavior

Another problem that is necessary for urban driving is the n-point turn, when the traffic lane is completely blocked or the vehicle needs to traverse other trajectories. The classic method for this behavior is simple:

1. Turn steering wheel to full left position and go forward until just before the vehicle hits any obstacle or crosses the traffic line.
2. Turn steering wheel to full right position and go backward until just before the vehicle hits any obstacle or crosses the traffic line.
3. If necessary, repeat 1 and 2 steps until the vehicle can drive on the desired path without hitting any obstacle or crossing any line.

However, there could be some situations that can be done with less steering wheel operations, so the proposed method is to use the RRT algorithm and the configuration space is just a simple circle that is the blue circle in Figure 4-30. The green points are new waypoints that the vehicle needs to pass, so a random tree is generated during max searching time or until it finds a proper path to move the vehicle to a new waypoint. If it cannot find the proper path, it acquires a node that has a minimum orientation different from the desire heading and the algorithm decides one case of four cases (shown in Figure 4-31) using the node point.

In cases 1, 2, and 3, the tree cannot find a proper path to move the vehicle to any waypoints, since a road with two traffic lanes does not have enough space for the vehicle to turn around. Therefore, the vehicle needs to follow the path to get to the final

node that has a minimum orientation difference from the desired heading. When the vehicle gets to that point, the algorithm builds the tree again and moves the vehicle. This process repeats until the vehicle can drive on the new path.

In case 4, the tree finds a proper path to the waypoints, so, by following the path, the vehicle can change its heading to the desired orientation.

This chapter described the specific implementation of the new sensing, storage, and path planning method for an unmanned ground vehicle tasked with navigating dynamic urban environments. The next chapter describes the testing procedure and analyzes and discusses the results of the various tests conducted.

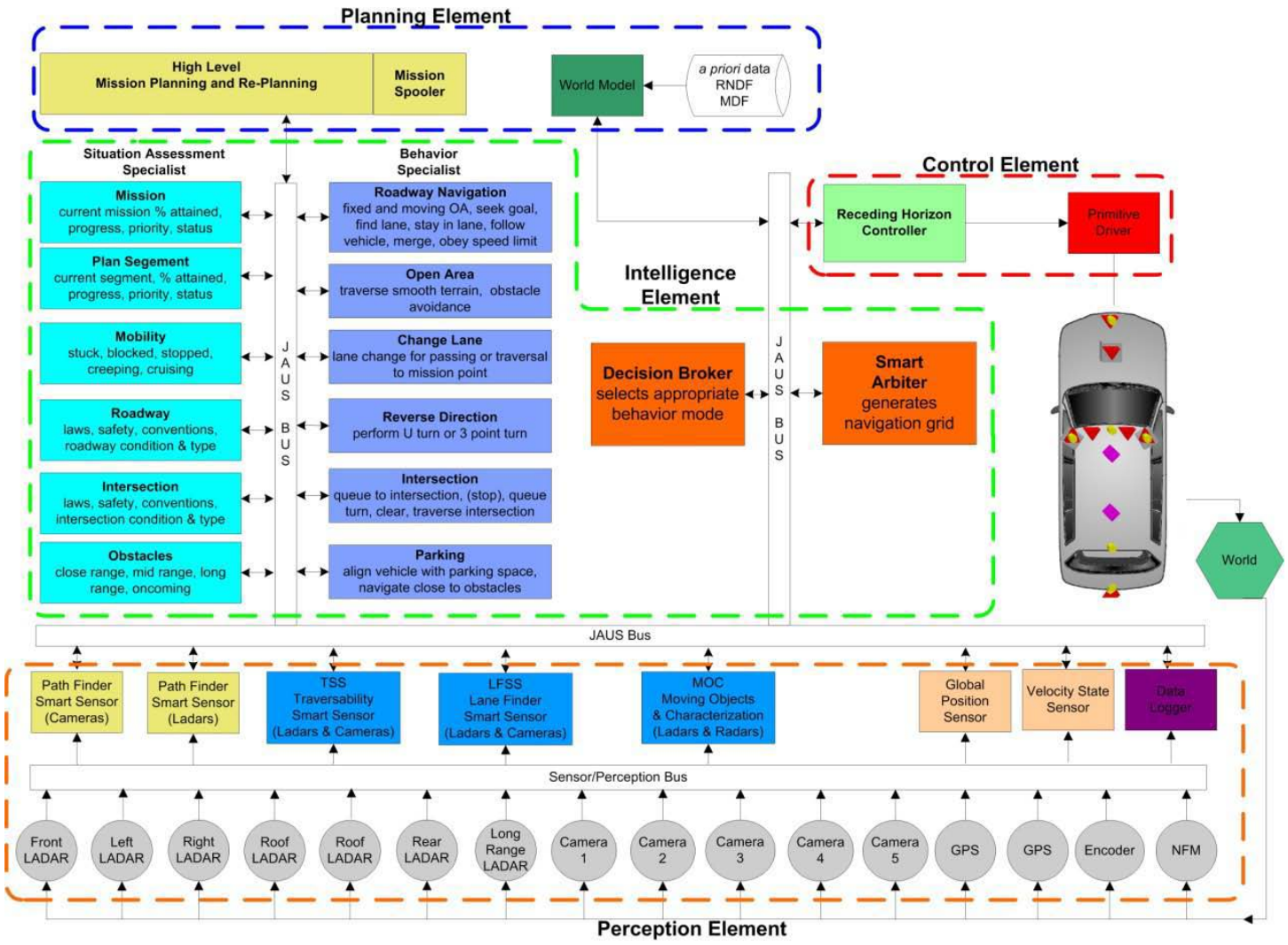


Figure 4-1. Urban NaviGator system architecture

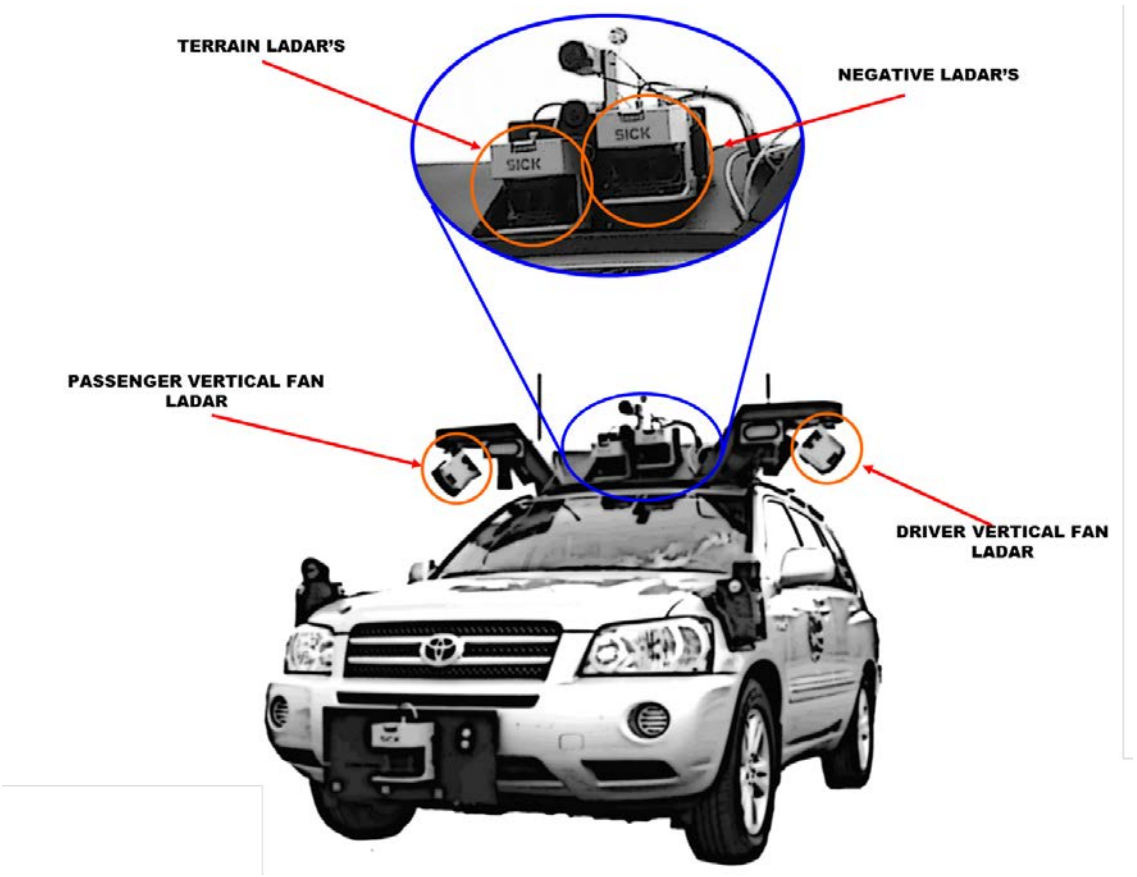


Figure 4-2. Picture of LADAR Sensors Layout on the Urban NaviGator that was taken by Jihyun Yoon.

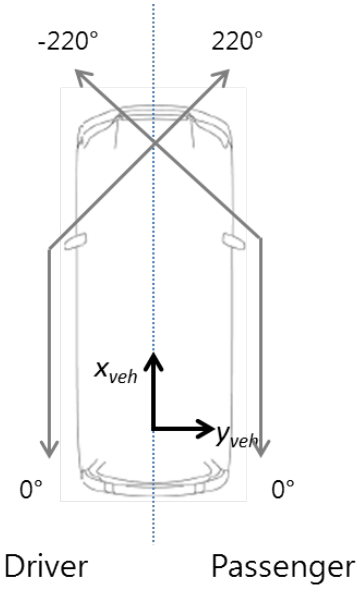


Figure 4-3. VFs Layout and Vehicle Coordinate Frame

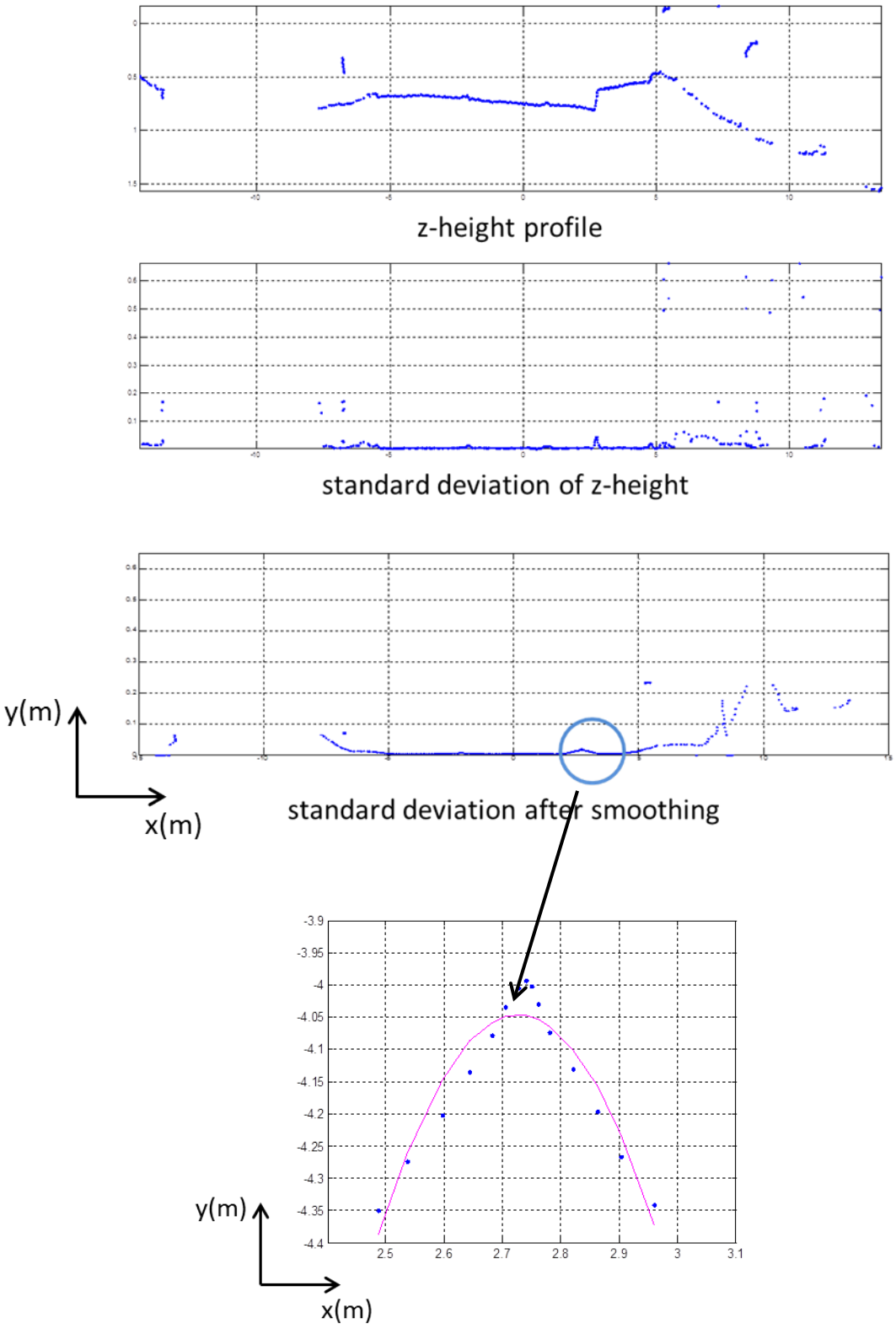


Figure 4-4. Peak Detection Procedures

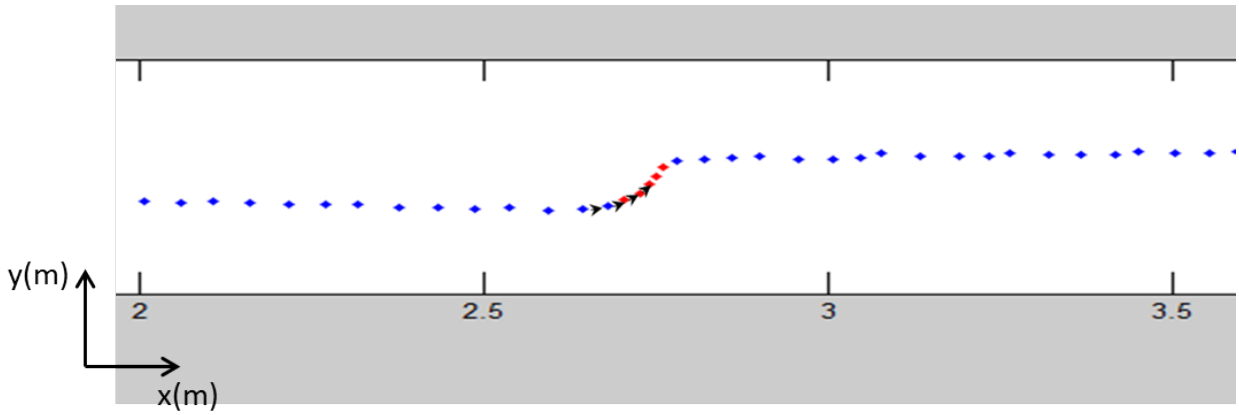


Figure 4-5. Slope Variance method

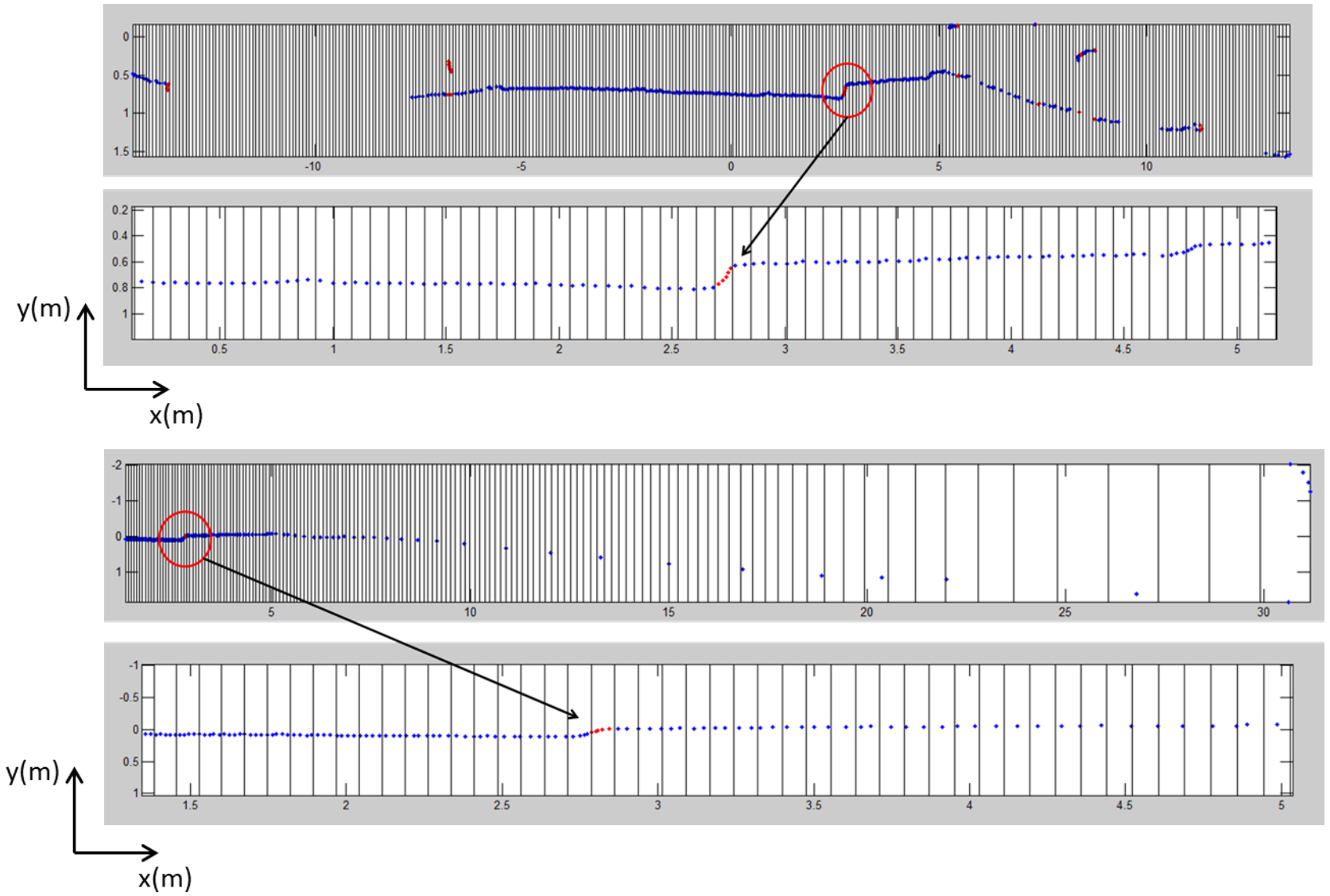


Figure 4-6. Grid-based method

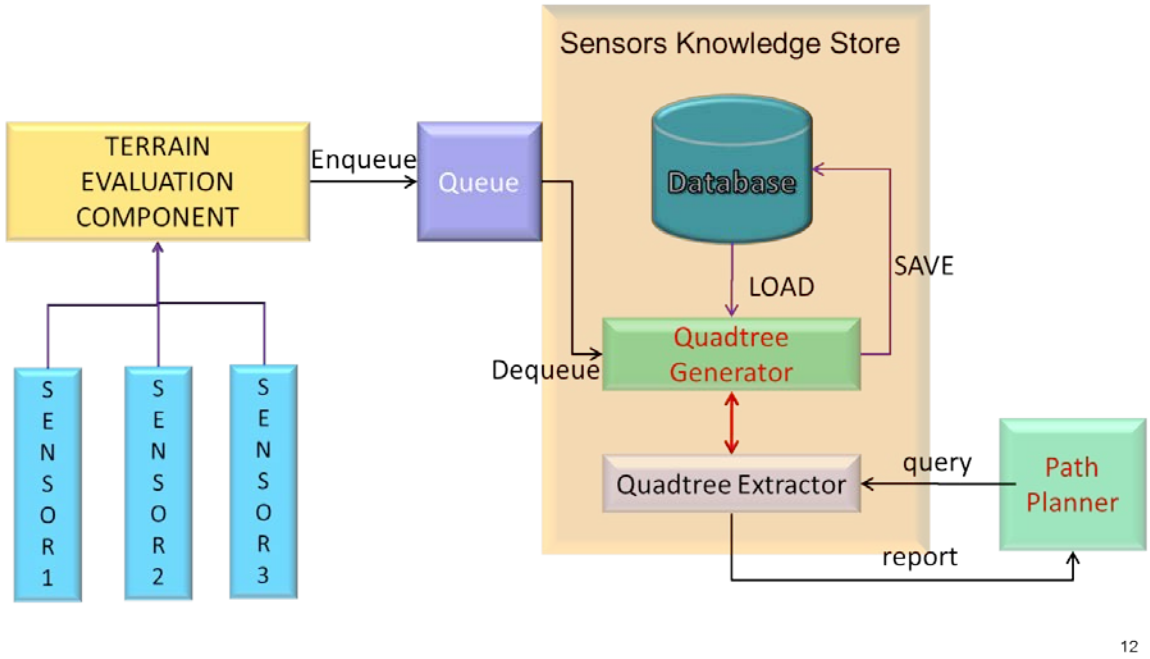


Figure 4-7. Quadtree Sensor Knowledge Store Component

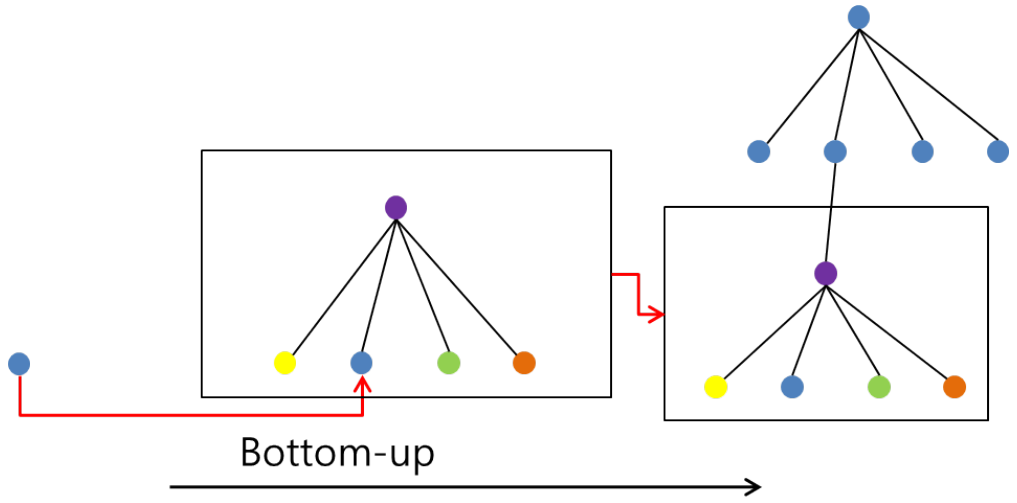


Figure 4-8. Bottom-up built Quadtree

Function : bottomupQuadtreeBuilding

```
1. Quadtree q;
2. get data from sensor and its position
3. if q is null
4.    q = makeNode(position,  data);
5.    make q root of the tree
6.    compute max range of the tree
7. else
8.    if position is inside of max range
9.     tempNode = q's root
10.      WHILE ( tempNode is not null and tempNode's level is not 0)
11.       index = findChild'sIndex
12.       tempNode = index'th child of TempNode
13.      endWHILE
14.      if tempNode's level is 0
15.         put the data into leaf node
16.      else
17.          make the rest nodes and branches from under the tempNode to leaf node put the
                    data into leaf node
18.      endif
19.    else
20.      compute new max range
21.      tempQ = makeNewRoot(new max range);
22.      bind q and tempQ
23.      make tempQ new Root
24.      make the rest nodes and branches from under the new root to leaf node
25.      put the data into leaf node
26.    endif
27. endif
```

Figure 4-9. Algorithm for Bottom-up Quadtree Building

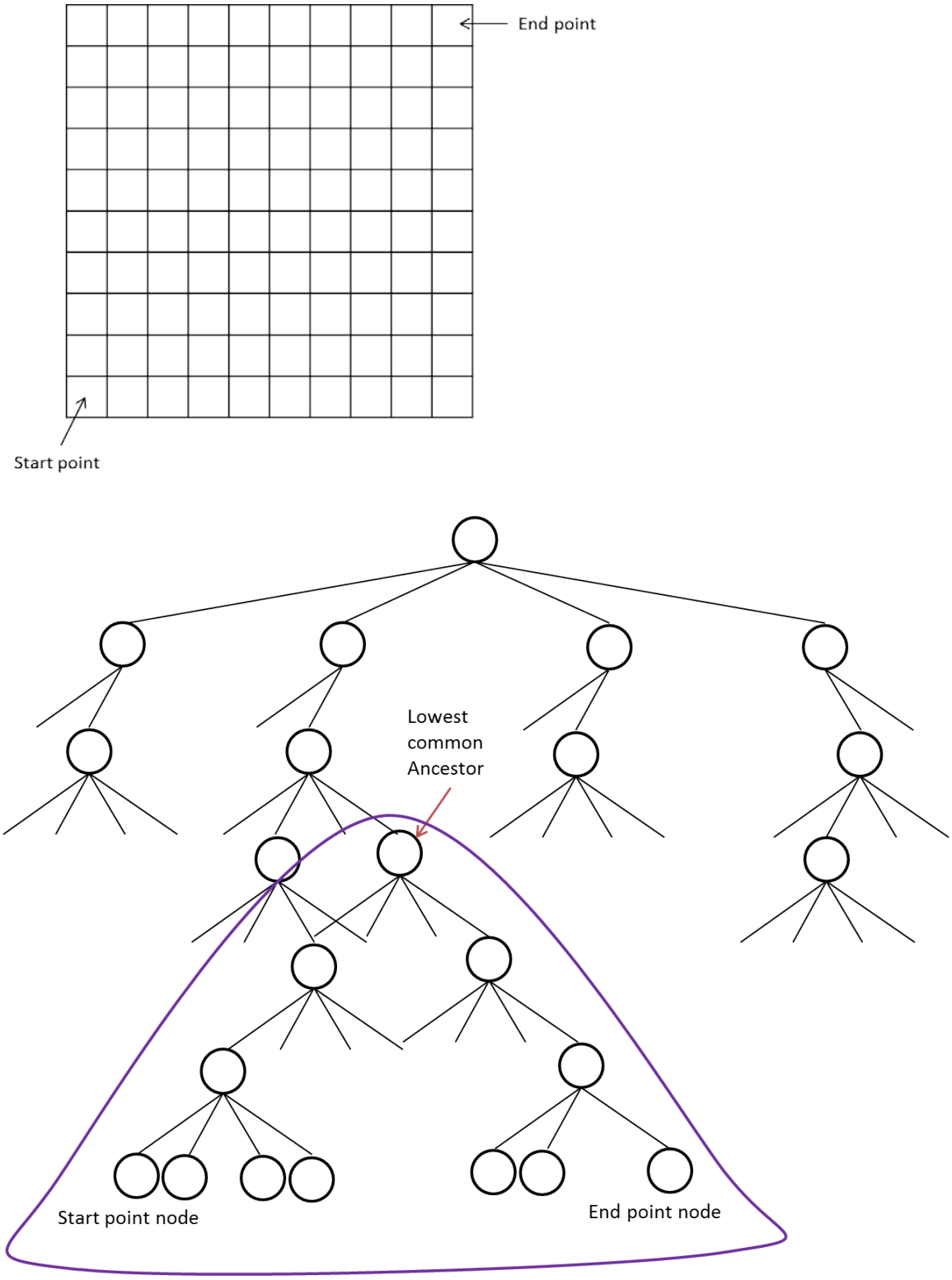
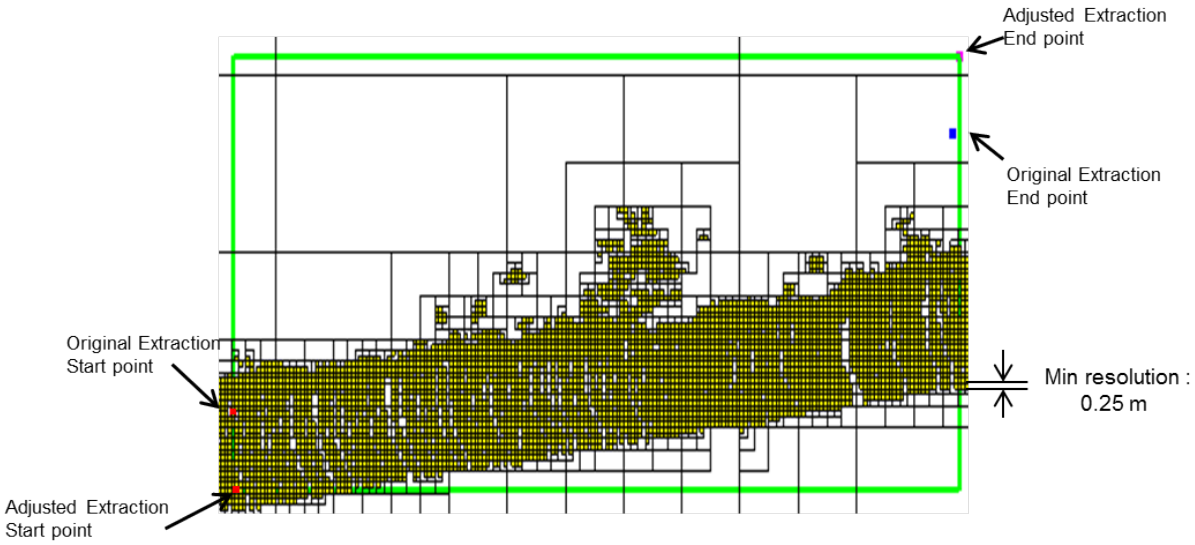


Figure 4-10. Quadtree Extraction



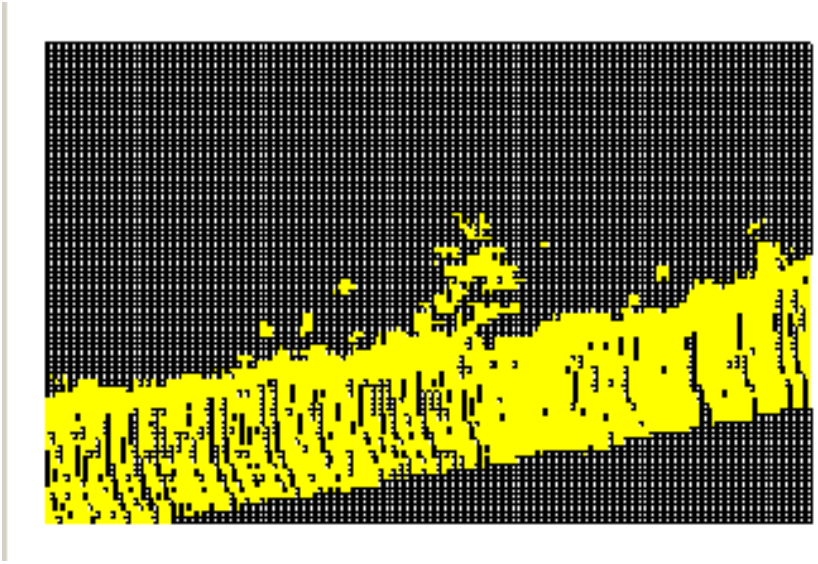


Figure 4-11. Quadtree Extraction example

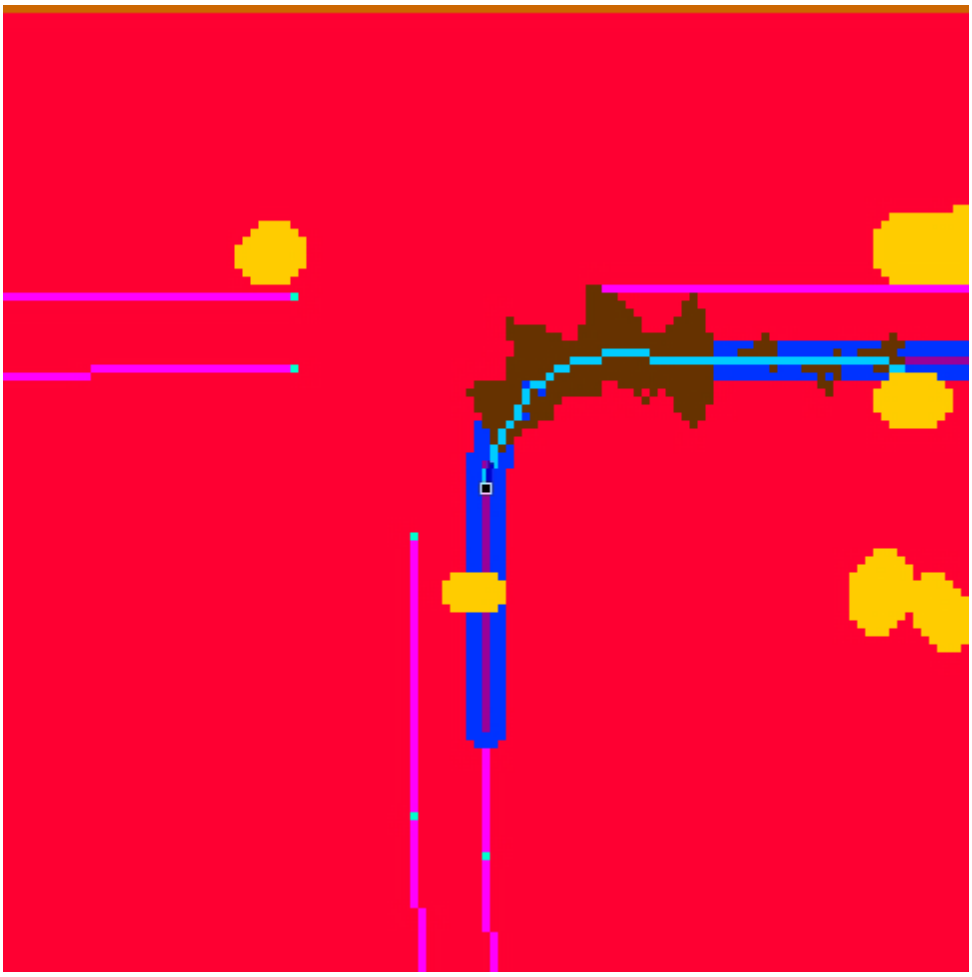
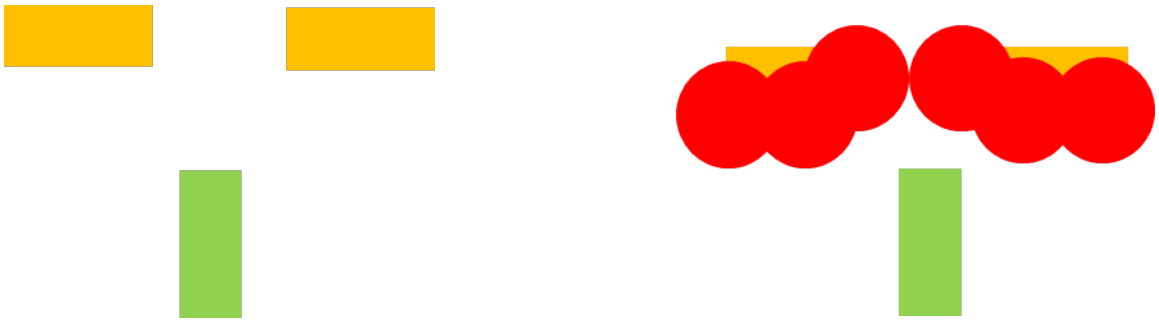


Figure 4-12. Traditional occupied grid method of CIMAR



(a) (b)

Figure 4-13. Example of problem of traditional occupied grid method

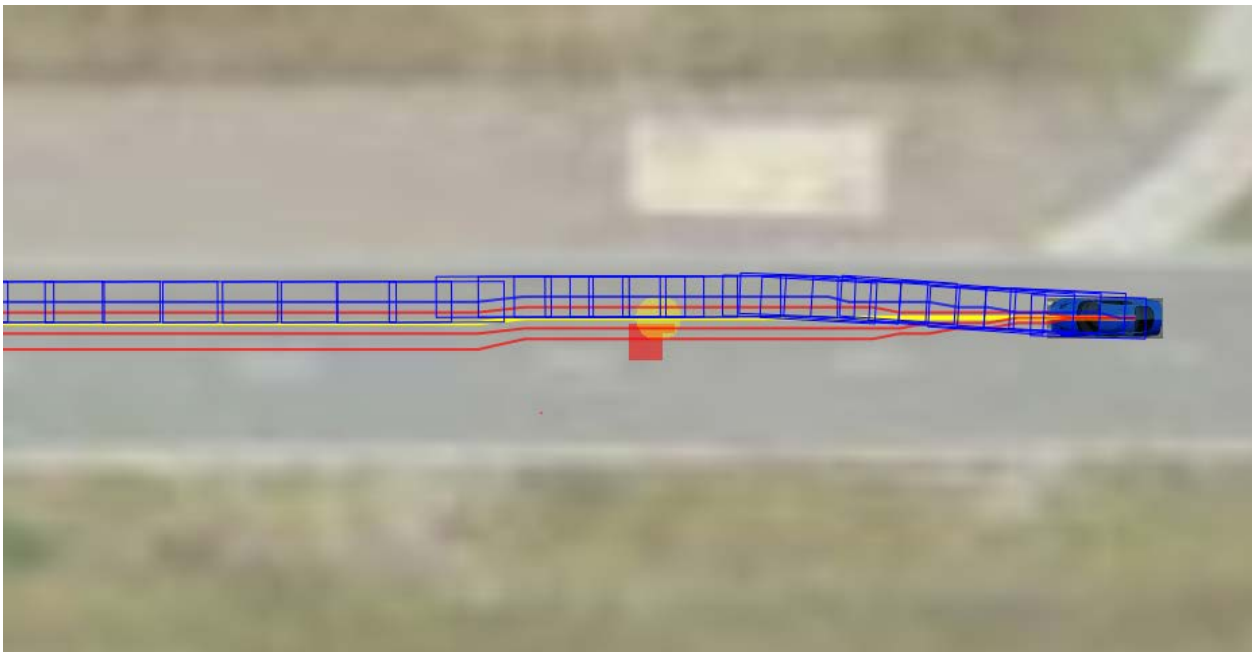


Figure 4-14. New occupied grid approach

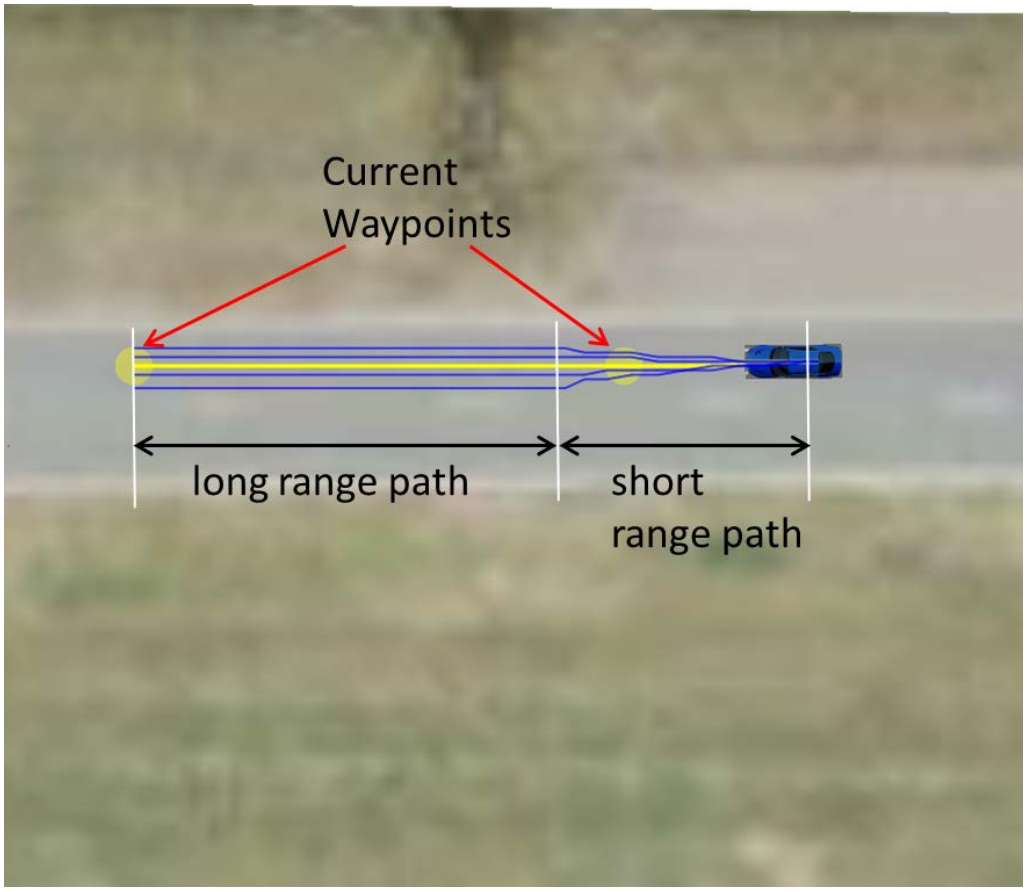


Figure 4-15. Vector-based roadway navigation path planner

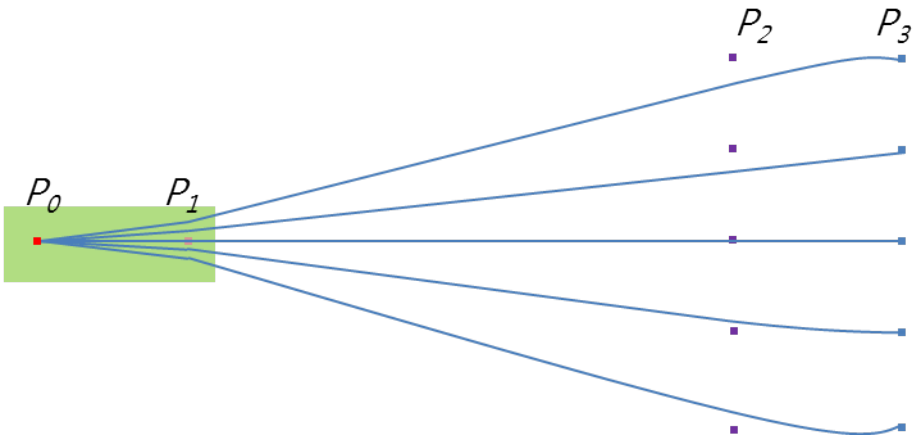


Figure 4-16. Control points selection for short range paths

Figure 4-17. Predefined speed profile of selected course on the Google satellite map

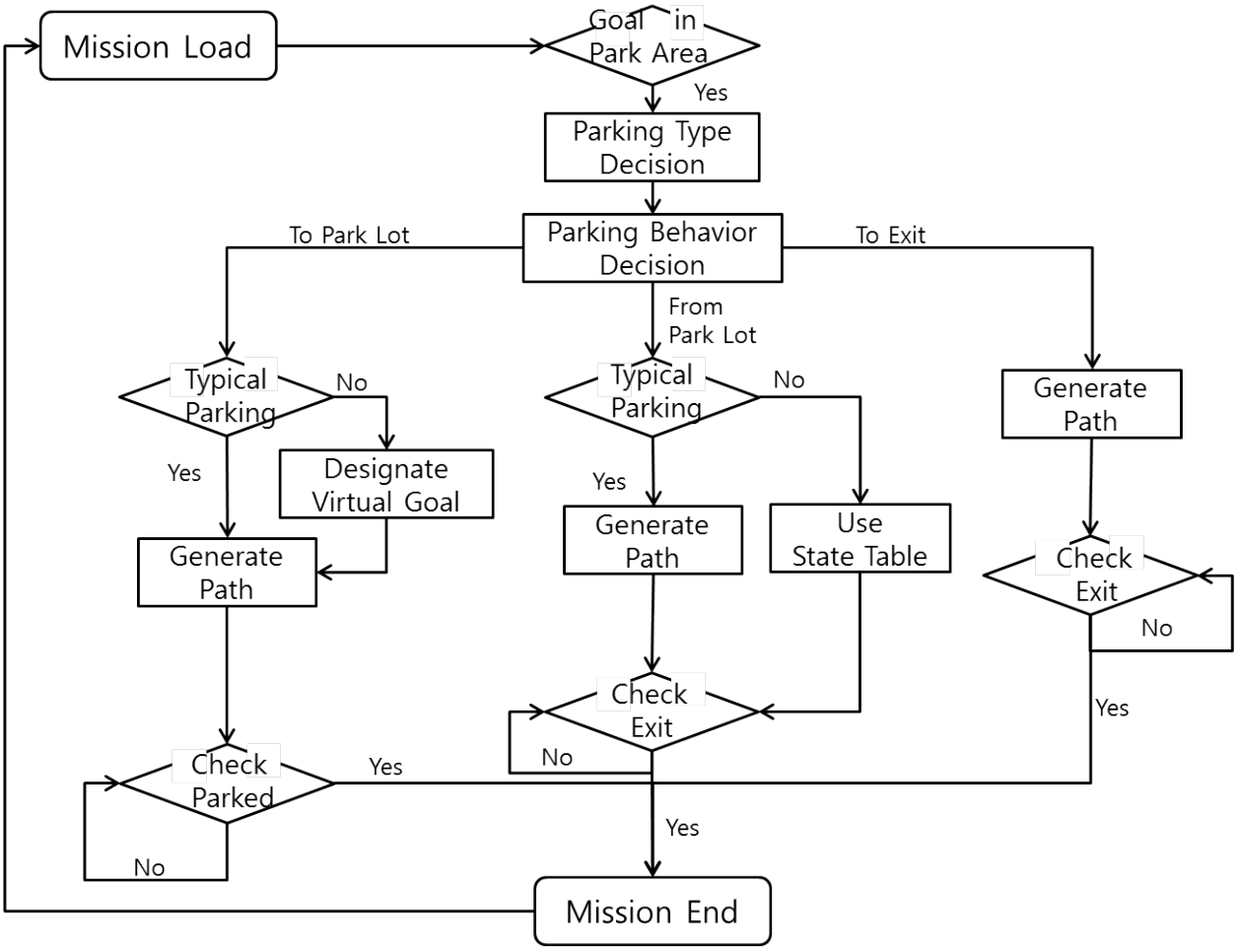


Figure 4-18. Parking process flow chart

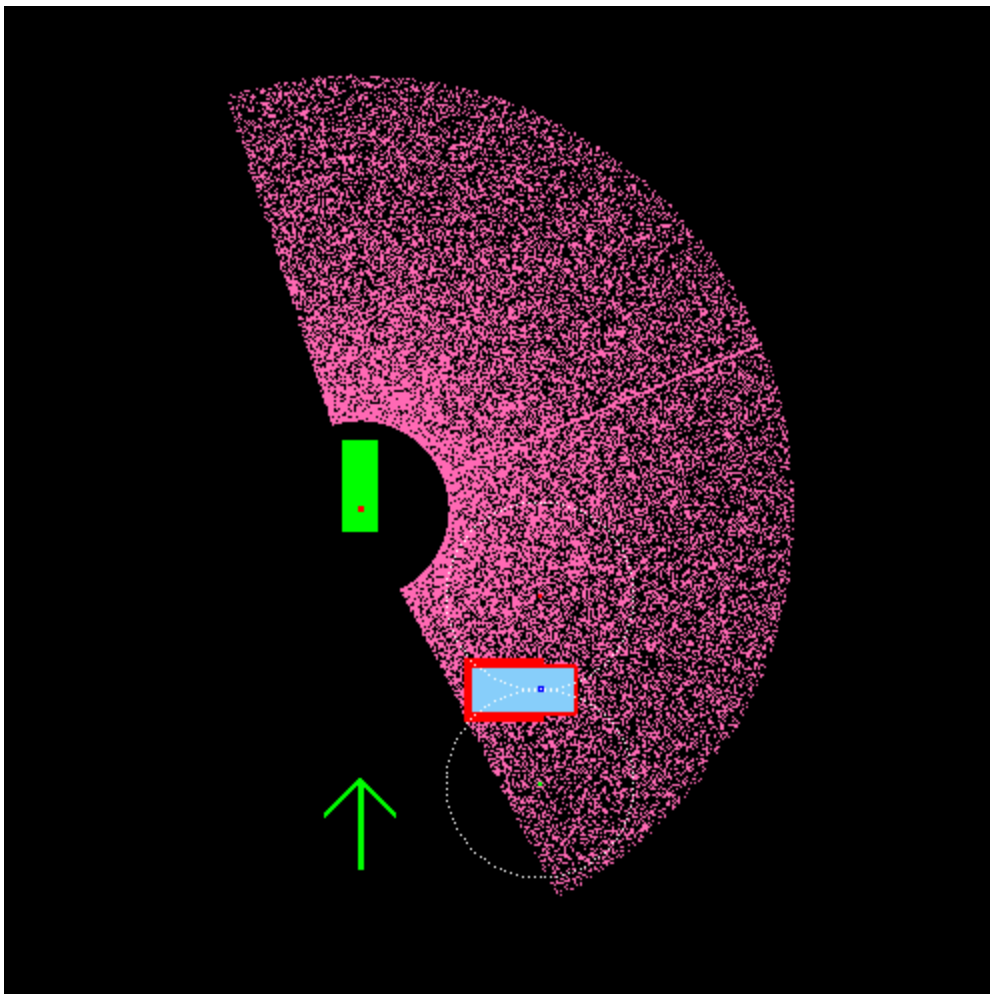


Figure 4-19. Configuration space for RRT algorithm

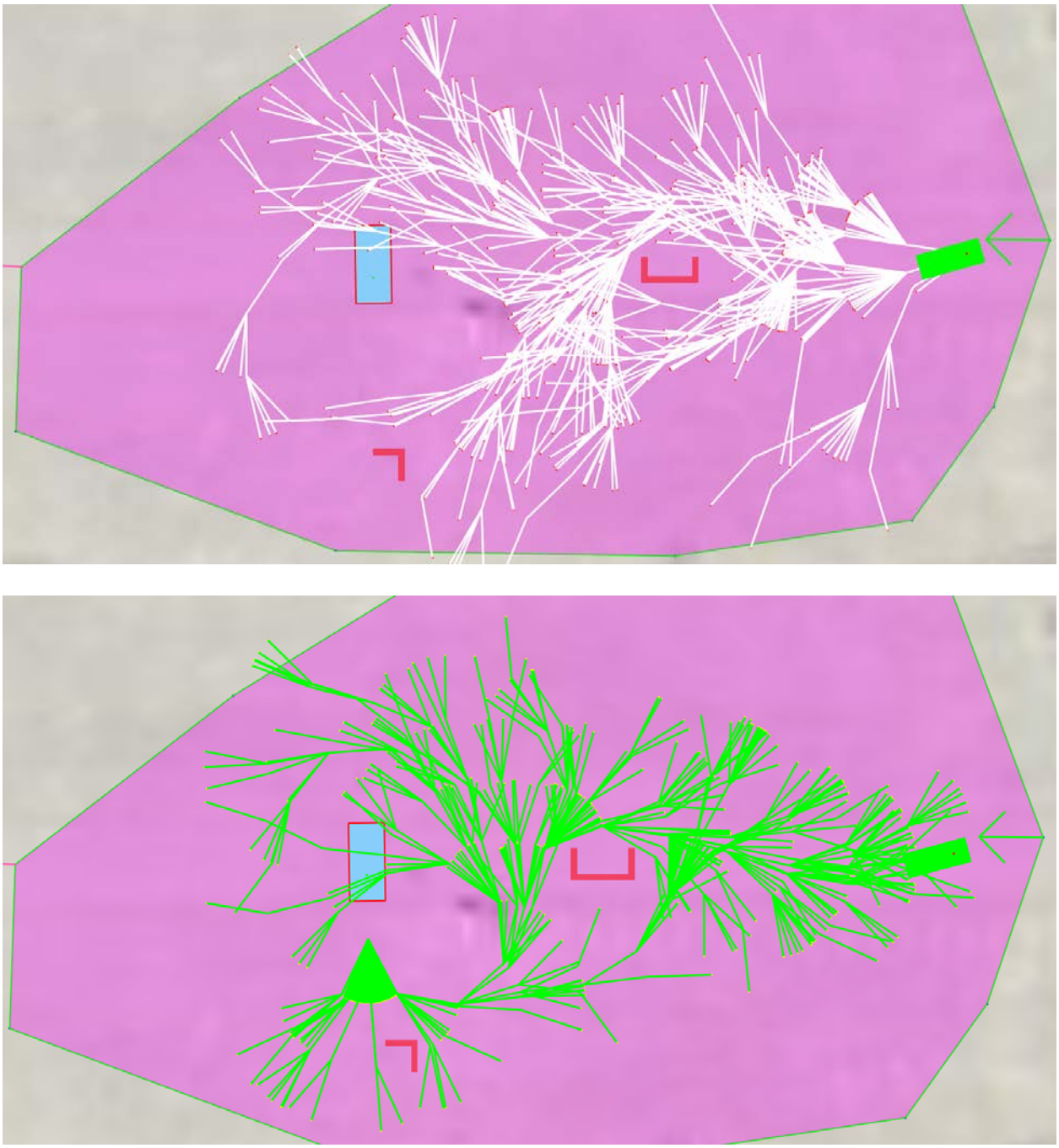


Figure 4-20. Tree expansion examples

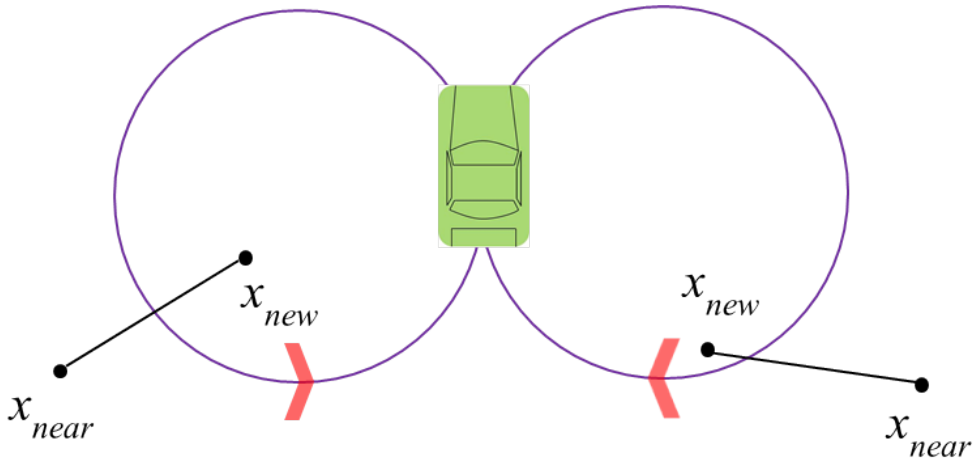


Figure 4-21. Minimum radius circle reach condition

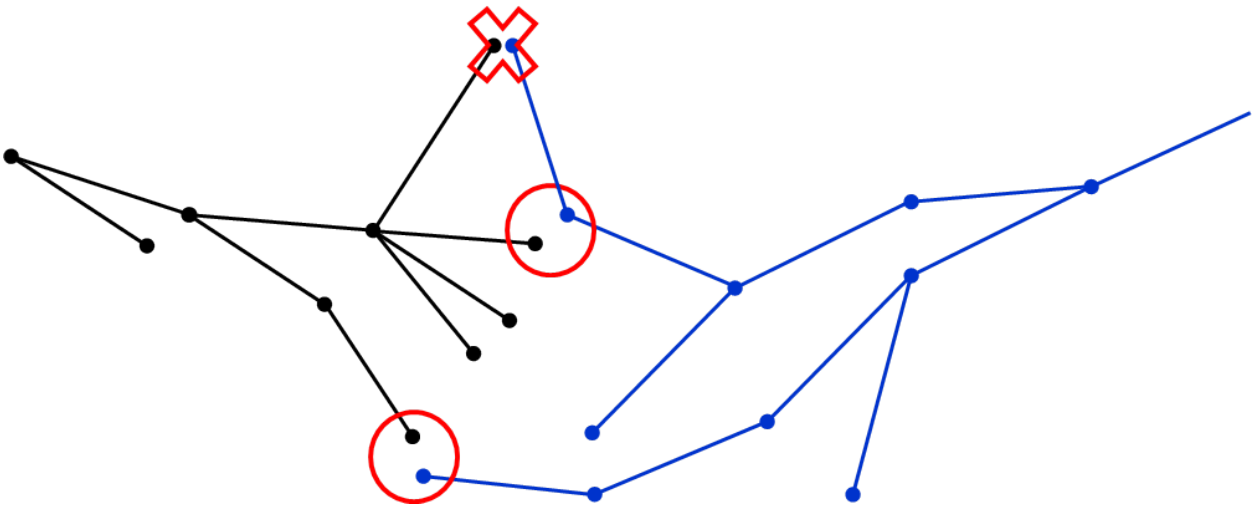


Figure 4-22. Example of tree connection

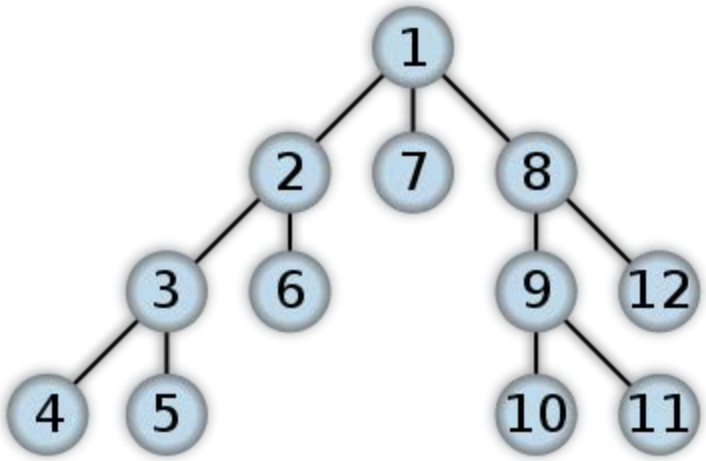


Figure 4-23. Depth-first search (DFS)

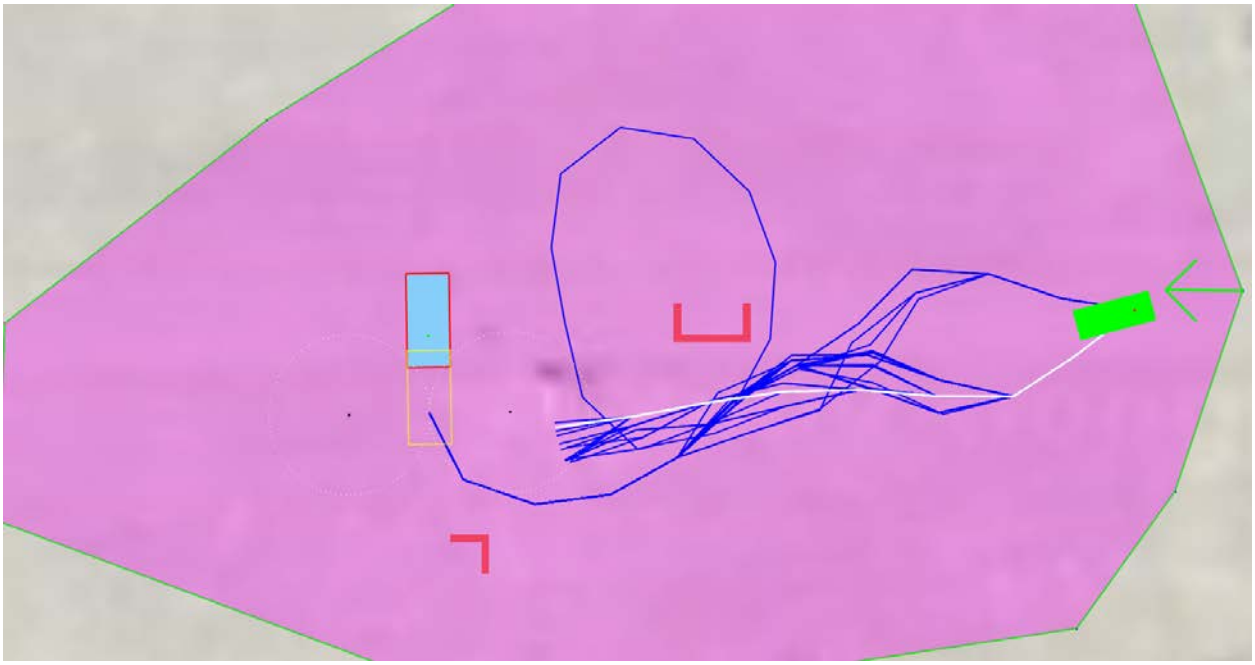


Figure 4-24. Candidate paths

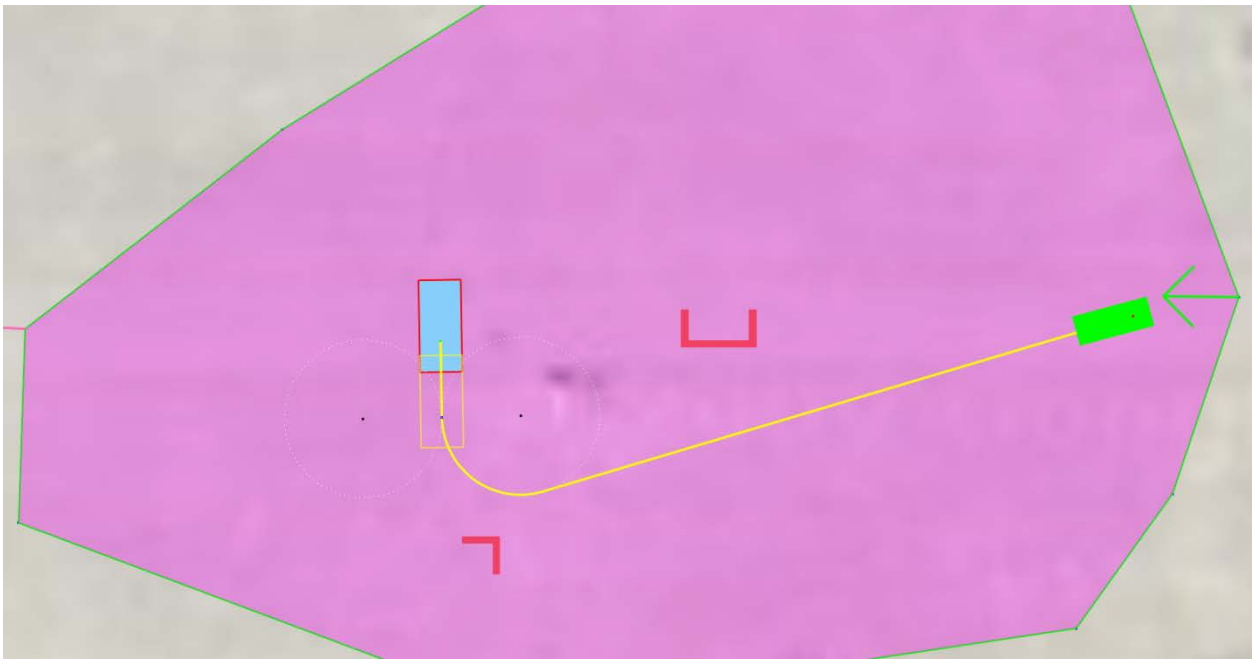


Figure 4-25. Final path

Function : SmoothPath (*path*)

```
1.  i ← 0
2.  j ← last index of the path
3.  index ← 0
4.  new path array[index] ← i
5.  index ← index + 1
6.  while i < j
7.      if collision free from path(i) to path(j) and curvature is smaller than threshold.
8.        then new path array[index] = i;
9.                i = j
10.               j ← last index of the path
11.               index ← index + 1
12.   else
13.     then
14.               j ← j - 1
```

Figure 4-26. Smoothing algorithms

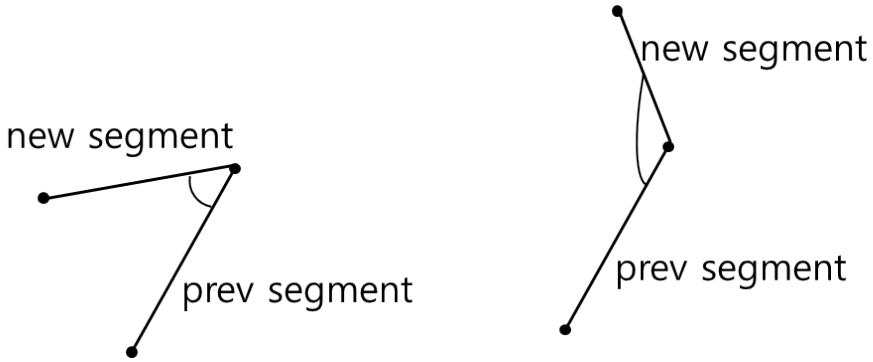


Figure 4-27. Requirements for Smooth Path

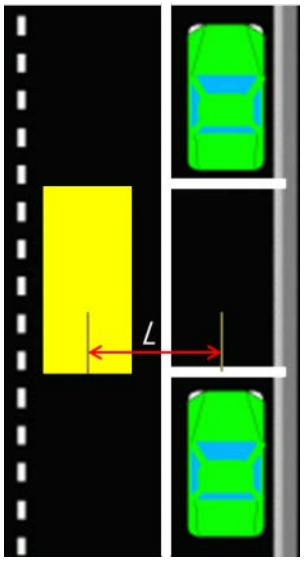


Figure 4-28. Parallel parking scenario

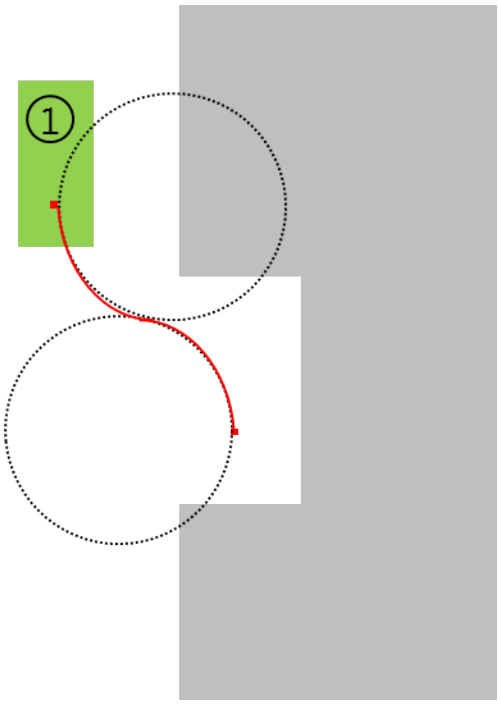


Figure 4-29. Parallel parking first step

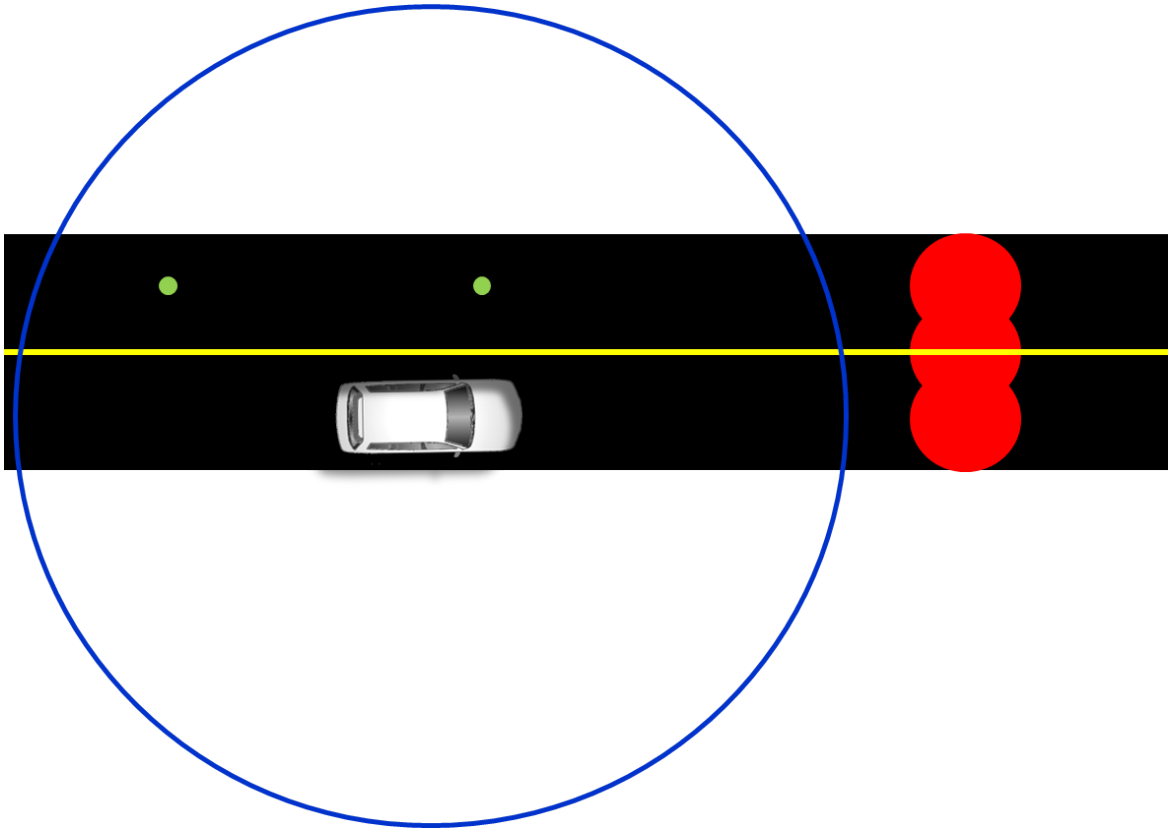
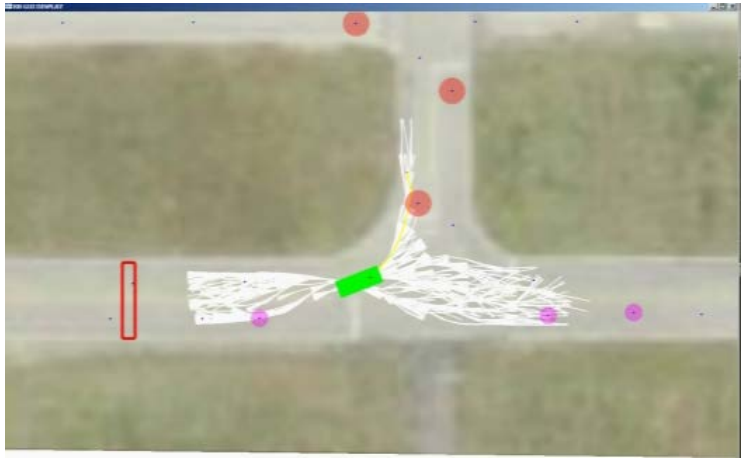
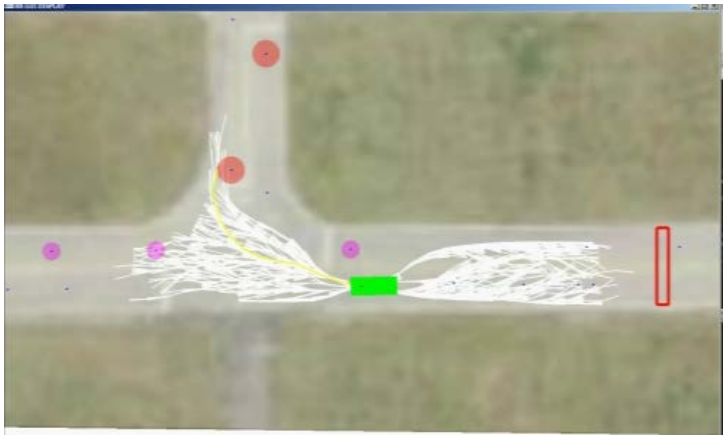


Figure 4-30. Configuration space for n-point turn

Case 1



Case 2



Case 3

Case 4

Figure 4-31. Four cases for n-point turn on the Google satellite map.

Table 4-1. Sensor Knowledge Store Data Structure

Bit	Content	Values
First 2 bits	Reserved	
3rd bits	Edge existence	0 : false 1 : true
4th bit	Waypoint existence	0 : false 1 : true
5th bit	Obstacle Existence	0 : false 1 : true
6th bit	Traffic line Existence	0 : false 1 : true
7 and 8th bits	Terrain Roughness	00 : untraversable 01 : rough 10 : smooth 11 : unknown

Table 4-2. Messages between components

TSS to SKS	Path Planner to SKS	SKS to Path Planner
Grid map size	Query ID	Report ID
Grid map resolution	Start query point	Adjusted start report point
Grid map center position	End query point	Adjusted end report point
Edge information grid map	Grid map resolution	Edge information report map
Height information grid map		Grid map resolution

CHAPTER 5
EXPERIMENTAL RESULTS

The research outlined in this document was implemented and tested extensively. Because of the complexity of maintaining a functional autonomous vehicle and the need for considerable resources in terms of manpower and facilities required to execute autonomous testing, testing of path planning was first conducted in simulation to verify the performance of the various developed algorithms. Testing the sensors' algorithm was done by human driving.

This chapter outlines the various testing methods and metrics used to measure the performance of these methods.

Test Plan

All the tests for the components were performed using a computer with an i7 core 1.70 GHz CPU, 4.0 GB memory, 1.0 GB video card memory, and 100 Mbps Ethernet communication.

Road Boundary Detection

The road boundary detection algorithm could be applied to the UGV for several of the operating behaviors encountered during the DUC. This test was performed with human drivers. Usually there are many road boundaries and objects in the urban areas, so the test of the algorithm focuses on detecting artificial road boundaries such as fences or curbs. To check validation of this algorithm in real time, the vehicle was driven at 20 mph, which is the speed limit in test area, the LADARs scanned the environment, and the algorithm tried to detect the road boundaries with the point clouds transmitted from the sensors.

Figure 5-1 is an aerial map of the University of Florida campus and the red arrows represent the route the vehicle drove. The results acquired from the areas in the yellow boxes were analyzed and compared by type of LADAR, type of method, and area, since the areas have continuous road boundaries.

Sensor Knowledge Store

This component is a new component for the UGV of CIMAR, so the important aspects that are needed to verify are how effectively the information is saved and how fast the search time is. In addition, the time to receive the information from the other component and send the extracted information to the other component is also verified.

This test was also done with the test road boundary detection algorithm since the component evaluates the environment of the vehicle and sends the information to this component immediately. This test was performed with a human driving.

After the test was done, the result with the number of nodes and amount of memory used was analyzed.

Path Planner

The tests of new the path planner algorithm for roadway navigation behavior, typical and parallel parking behaviors, and n-point turn behavior were performed by simulation with the RNDF acquired from the test site at the Gainesville Raceway (Figure 5-2). This site has served as the test facility for many of CIMAR's autonomous ground vehicle projects. The site provides paved road segments used to test many of the required behaviors for the DUC, including straight and curved road segments with multiple lanes, intersections, dead-end streets, and a parking area. Sections of this area were selected for simulating the required test environments. In the figure, blue points and red circles represent waypoints and stop signs, respectively.

The first step in the tests is to set waypoints that the vehicle has to pass and decide its type of behavior. Figure 5-3 shows the selected course in yellow line segments. The second step is to set virtual obstacles with a simulator built in C/C++ and OpenGL to verify that the generated path can make the vehicle able to avoid the obstacles by finding candidate paths and an optimal path.

Figures 5-4 and 5-5 show close-up visualizations of the waypoints defining the road segment used for the parking and parallel parking tests. In those figures, the polygon in light purple color is open area, the green rectangle is vehicle, the blue rectangle is parking spot, the green arrow is entrance to parking area, and the pink arrow is exit from parking area.

Test Metrics

A number of different performance measures were selected to be recorded to evaluate the new methods described in this dissertation. These metrics served to investigate the effectiveness of the new methods in allowing a robotic system to navigate in urban areas.

The first set of metrics is associated with how well the road boundaries detection algorithm can find the road boundaries depending on the sensor and method. It was found that the rate of detecting the curbs depends on the sensor type and curbs type.

The second set of metrics is the memory usage and Quadtree building time complexity of the Sensor Knowledge Store and this result is compared with the storage in the classic grid map.

The last set is the search rate of the path planner. For the roadway navigation behavior, the update rate of the path planning loop is found. For the parking and parallel parking behaviors, the number of nodes and update rate of the RRT algorithm is

recorded, depending on the distance to the goal point. In addition, the error between the heading of the vehicle and the desired orientation of the parking spot is recorded.

All these values were logged in a separate thread by the corresponding component so that it did not interfere with the performance of the rest of the algorithm. The resulting data was recorded, plotted, and analyzed to determine the results of each of the tests.

Results

Road Boundary Detection

The test for this component was done by a human driving on the University of Florida campus. The original goal of this component was to find artificial or natural boundaries of the traffic road and assign the probability that the boundary exists to the proper cell of the grid map. Once the detection process is done, the final step is to adjust a current traversability value of each cell depending on the probability of a road boundary. Figure 5-6 shows the edge grid map resulting from area1 of Figure 5-1 and the color table, depending on the traversability values. Except for unknown cells, the initial values for cells was 32 a neutral value. If a cell has 0% probability for edge existence, as would be true for a road, the cell retains the neutral value. On the contrary, if it has 100% probability, which means that all four sensors detect an edge at the same point, the edge value of the cell changes to 2, which is the lowest available traversability value of the grid map.

Figure 5-7 compares the classic grid map with the grid map that has the traversability values adjusted, depending on the probability of the edge existence. The upper figure shows the classic grid map by cell-base evaluation and the lower figure is based on the grid map with the new method. In this area, the vehicle was driving to

South (downward in the figure) and the vehicle is on the center cell of the grid map. As previously mentioned, the traditional grid-base method was not enough to distinguish road boundaries in urban areas, which means the boundaries can be detected by lower traversability values than a smooth road, but the cells are still drivable areas. The traversability values of cells having the road boundaries could be lowered by the confidence value estimated using the new method.

Table 5-1 shows the update rate of this component, depending on the then-current situation. Intuitively, a faster update rate of the sensor algorithm could give the other components a better chance to prepare their own action using the information, but the desire update rate is set as 40.00 Hz to prevent wasting memory usage. When the component only evaluates the vehicle's environment, the component can run the fastest but this does not make sense since the component has to send the information to other components. Therefore, the update rate, when it sends all the messages, has to be considered for performance evaluation. The result was 33.85 Hz when it sent all the messages to only one other component, which was obviously slower than the desired update rate speed, but enough for the current autonomous driving car that runs at 30 mph.

Table 5-2 through Table 5-7 show the final results acquired from area1 through area6 of Figure 5-1 using three methods and four sensors in real-time testing. A sensor, Terrain LADAR, provided the best results since it is fixed to see ahead of the vehicle and evaluates terrain at a shorter distance, and finer resolution than the other sensors. Another sensor, Driver Vertical Fan, returned the worst results, since it was rotating

using an attached motor and usually the road boundaries of the lane where the vehicle was driving were on the passenger side.

From the results, the shorter the beam distance and the smaller the angle between the beams, the better the estimation that could be provided.

Sensor Knowledge Store

The testing for this component was performed using a TSS component test where the component sent the edges' information, terrain data, and obstacle information to the Sensor Knowledge Store component via a JAUS message. Once this SKS received the message, it unpacked it by following the rules for the message and then the unpacked information is saved to the Quadtree.

Figure 5-8 shows the resulting image of the Quadtree using OpenGL software to render the visualization. The thick black line (actually they are a series of squares with 0.25 × 0.25 m resolution) shows the path of the vehicle similar to that of Figure 5-1. As shown in this figure, the upper parts and lower right parts of the center are not estimated, but the parts are generated automatically since some of the lower left part is evaluated.

Figure 5-9 shows the local image when the vehicle was driving around area2 of Figure 5-1 and was driving to the West. In the figure, only the cells with the lowest resolution, 0.25 x 0.25 m in this approach, are painted. The yellow cells are unknown areas. The green cells and pink cells represent smooth and rough terrain, respectively. The blue cells and red cells represent cells with road boundaries and cells with obstacles that are non-drivable areas, respectively.

Figure 5-10 is an array extracted from the same area as Figure 5-9 and the colors that are assigned to each cell represent the same characters as in Figure 5-9, but the

nodes that are internal nodes are converted to specified resolution cell and are given unknown traversability values. Once the converting process is done, the array is sent with specific messages to the other components.

Table 5-8 shows the resulting parameters obtained from the Quadtree. Of particular note is the “memory usage.” Even though the range of the map is quiet large, the size of the data is small enough to be saved. However, if the user does not pay considerable attention, the system could fail due to computer memory limit or hard disk storage limit. So the user has to consider which information is to be saved into the Quadtree and where the data will be stored, such as a hard disk drive or memory. In this research, the used memory for each node of the Quadtree is only 18 bytes at most since it only saves the pointers of four children (16 bytes at most), environmental data (1 byte), and scaled-height information (1 byte).

The update rate of the component is distinct from the uprate of Quadtree building, since the Quadtree building process is operated by a different thread. To verify the effectiveness of this approach, the usage memory of the Quadtree is compared with the fixed size grid map. The total range of this test was 8192 x 8192 m, so if the algorithm uses a fixed grid map of 0.25 m resolution, the grid map has 32,786 rows and 32,786 columns. The algorithm saves 2 bytes of information into each cell, so it uses 2049.13 Mbytes. Thus the proposed algorithm requires a slightly longer time to search proper nodes for point clouds and extract necessary areas to arrays, but it can reduce the memory used dramatically.

From the results, it is verified that the Quadtree data structure is good enough to be used to save huge sensor information data to recall in future driving.

Path Planner

This component was tested by a simulator generated by C/C++ and OpenGL. The first test was roadway navigation behavior that happens most frequently in autonomous driving. To test this behavior, a course that is composed of a set of waypoints was selected and a set of the obstacles was placed to observe several situations:

1. The obstacle completely blocks a single traffic lane.
2. The obstacle partially blocks a single traffic lane.
3. The obstacle blocks multiple lanes completely.

The path planner needs to control the vehicle, depending on which obstacle is the case. Figure 5-11 is about case 1 where the single lane is completely blocked by the obstacle. In this case, the vehicle has to stop a safe distance from the obstacle (Figure 5-11 (a)), and check if the next lane is clear. If it is, the algorithm generates the proper paths and the vehicle follows the optimal path with lane change speed that is defined at 10 mph in this dissertation (Figure 5-11 (b)). In this figure, the black square on the original lane is the projected (not the true) vehicle position onto the original lane. Using the projected vehicle, the algorithm checks if the true vehicle passed the obstacle or not. If it passed, the algorithm generates paths to move back to the original lane (Figure 5-11 (c)). If the two vehicles are very close, the algorithm goes back to original algorithm to pass the defined waypoints (Figure 5-11 (d)).

Figure 5-12 shows the result of case 2. In this case, the vehicle can avoid the obstacle without changing lanes. Each of the generated paths has a cost and the paths that would hit the obstacle have much higher costs so the optimal path is sought. Once the optimal path is found, the vehicle can avoid the obstacle by following that path.

Figure 5-13 shows the results of case 3. In this case, the vehicle can change lanes without stopping when the current lane is completely blocked by obstacles. The procedure to do this is same as case 1, except the vehicle does not have to stop.

The next behavior that the planner has to consider is the parking behavior. As mentioned previously, the path for the parking behavior can be found using the RRT algorithm. Figure 5-14 shows the result of typical parking behavior. The vehicle approaches to an open area (Figure 5-14 (a)). When the vehicle gets to the area, the algorithm generates the path using the RRT algorithm and the vehicle follows the path (Figure 5-14 (b)). Once the vehicle is properly parked, the algorithm needs to make a temporary goal to get the vehicle out of the parking place (Figure 5-14 (c)). When the vehicle is out of the parking place completely, the algorithm generates new paths to move the vehicle to the exit of open area (Figure 5-14 (d)). Figure 5-15 shows the parametric results of the RRT algorithm. Figure 5-15 (a), (b), and (c) are the graphs of the number of nodes, the number of candidate paths, and searching time, depending on the distance between the current vehicle position and goal position, respectively. The results show that the parameters and the distance are two different things. The number of parameters is decided depending on the minimum searching time that initially is defined as 0.1 sec in this dissertation.

There is another type of parking behavior, parallel parking. Figure 5-16 shows the result of the behavior. Figures 5-16 (a) and (b) are the same procedures as a typical parking place, but the goal for this behavior is a virtual parking lot in a lateral position to the original goal point. Once the vehicle is in the virtual parking space position, the algorithm starts moving the vehicle (Figures 5-16 (c) and (d)) in the way that was

described in Chapter 4. Figures 5-16 (e) and (f) are the same steps as described at typical parking behavior.

The last path planning test performed in this dissertation is n-point turn behavior. The four cases mentioned in Chapter 4 are shown in Figure 5-17 through Figure 5-20 respectively.

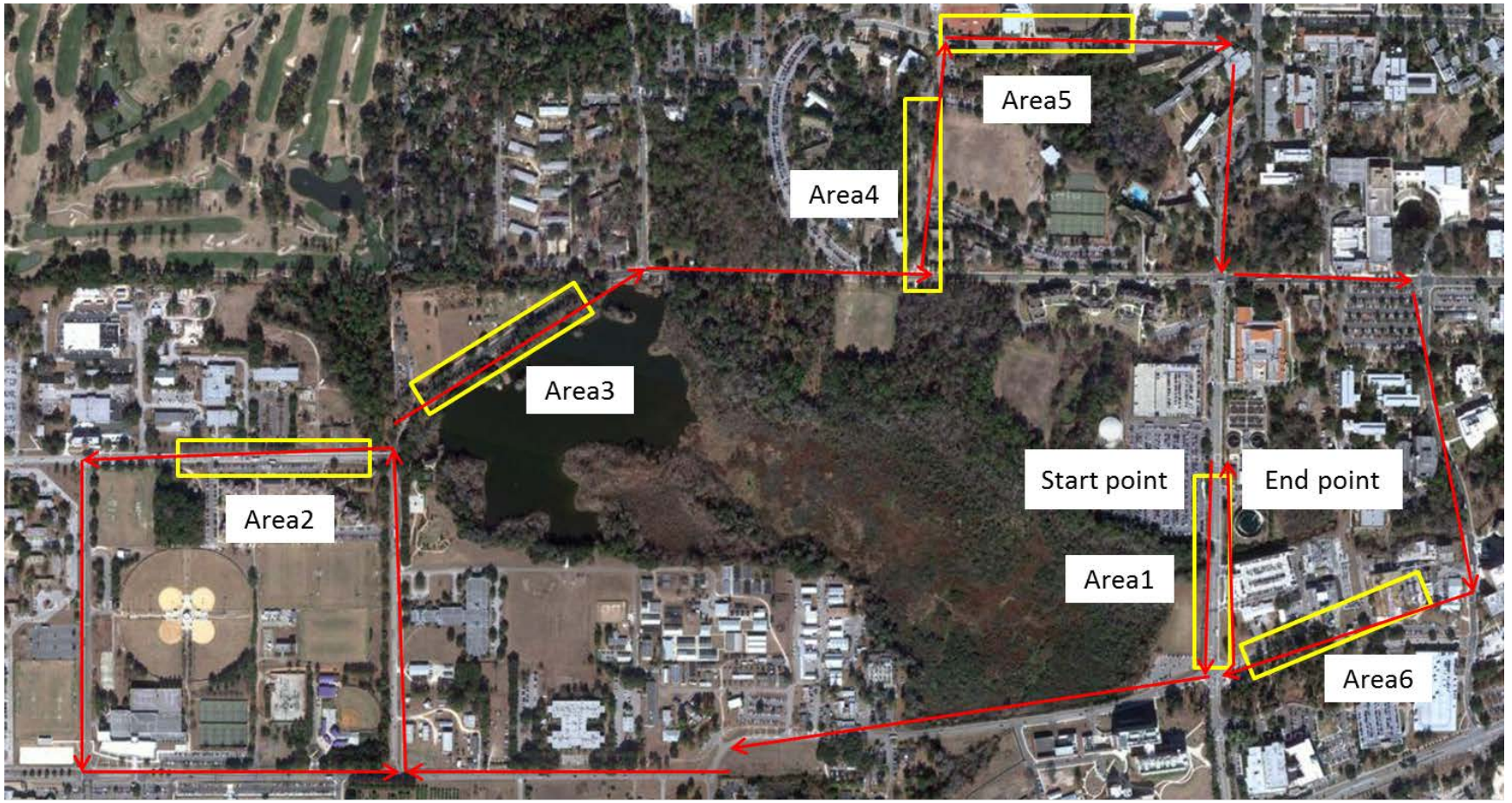


Figure 5-1. The University of Florida campus test area taken from the Google map.

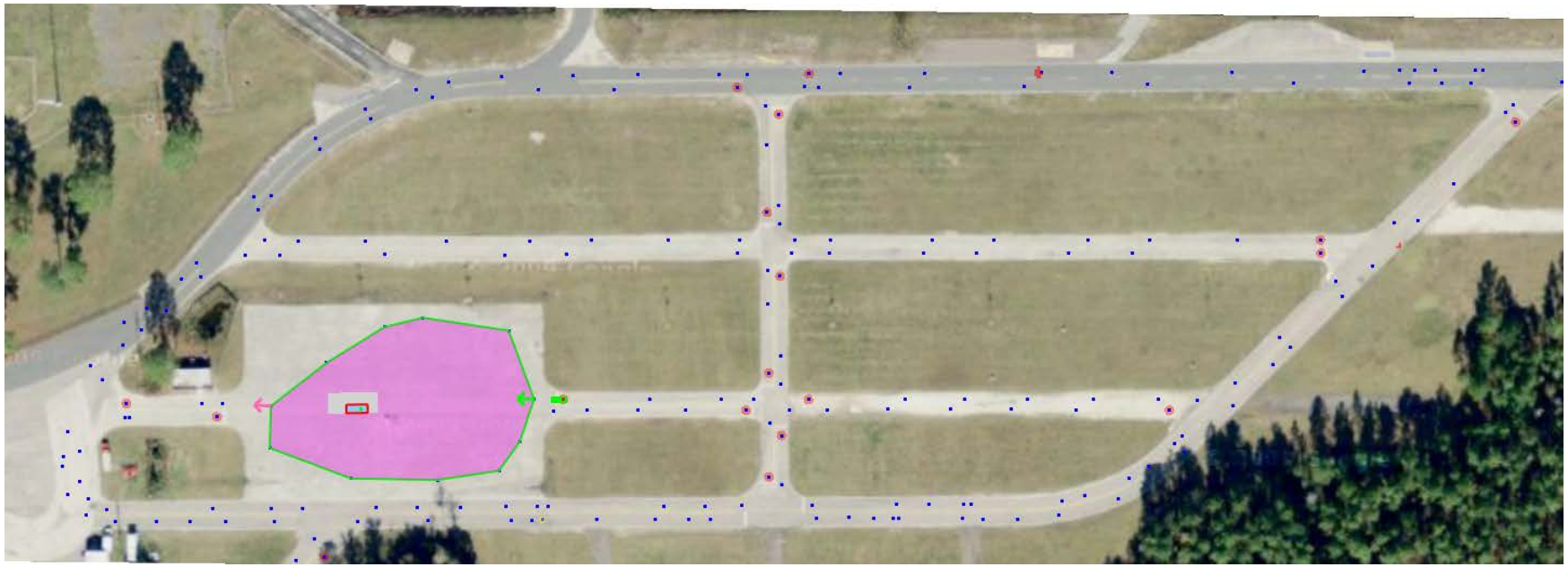


Figure 5-2. Gainesville Raceway pit area with points defining test areas on the Google satellite map.

Figure 5-3. Selected course on the site for roadway navigation behavior on the Google satellite map.

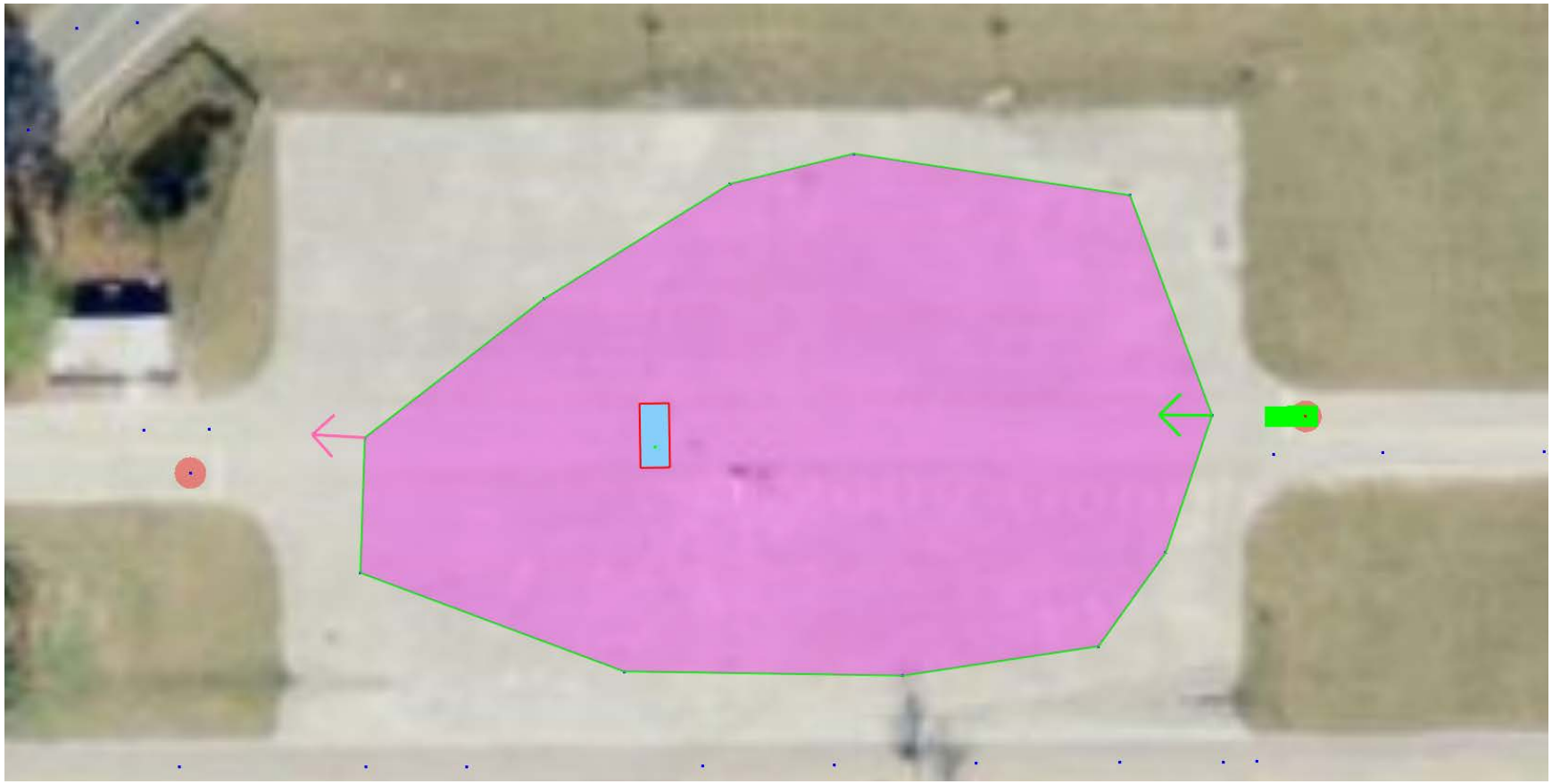


Figure 5-4. Gainesville Raceway pit area for parking behavior on the Google satellite map.

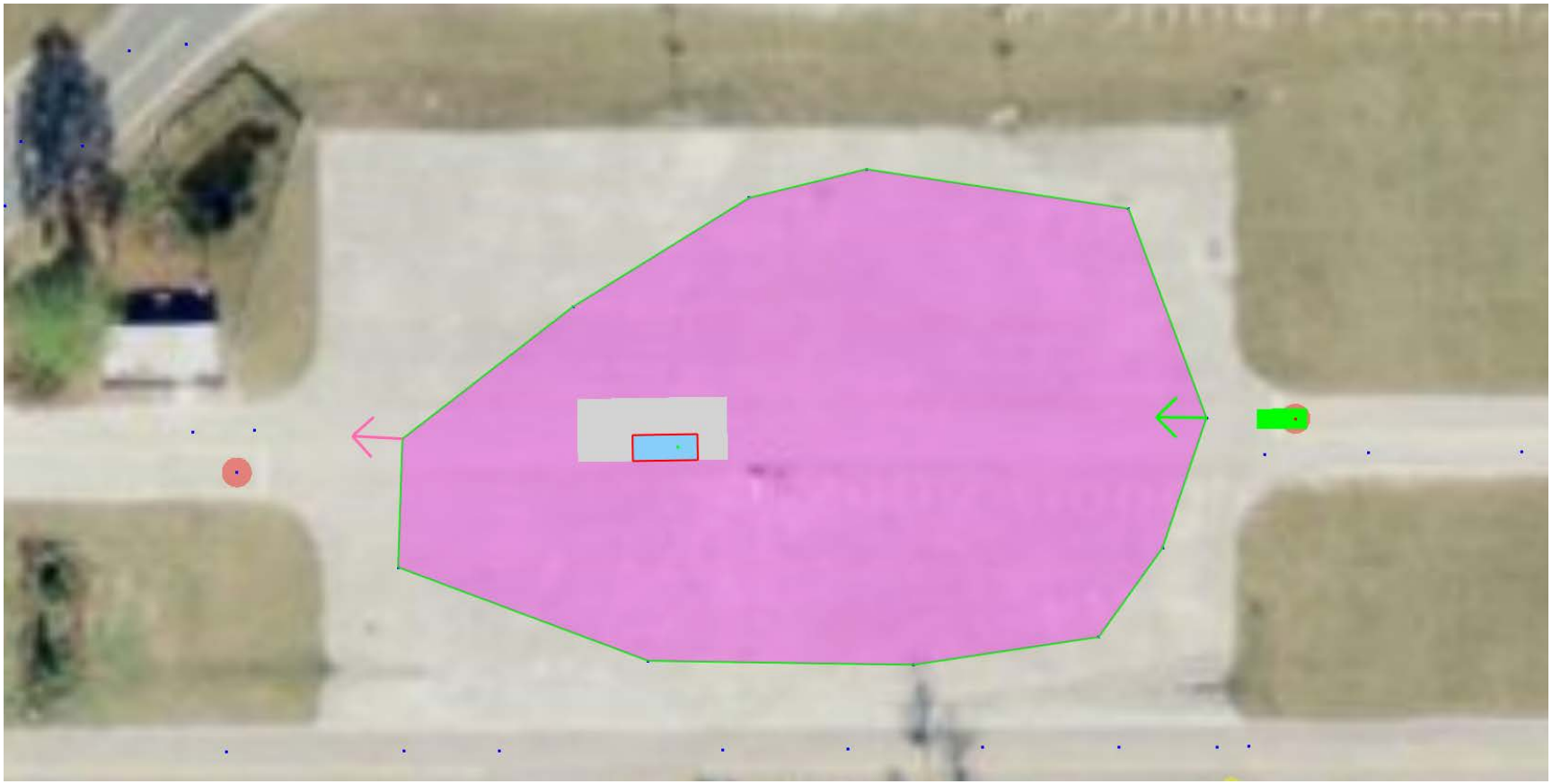
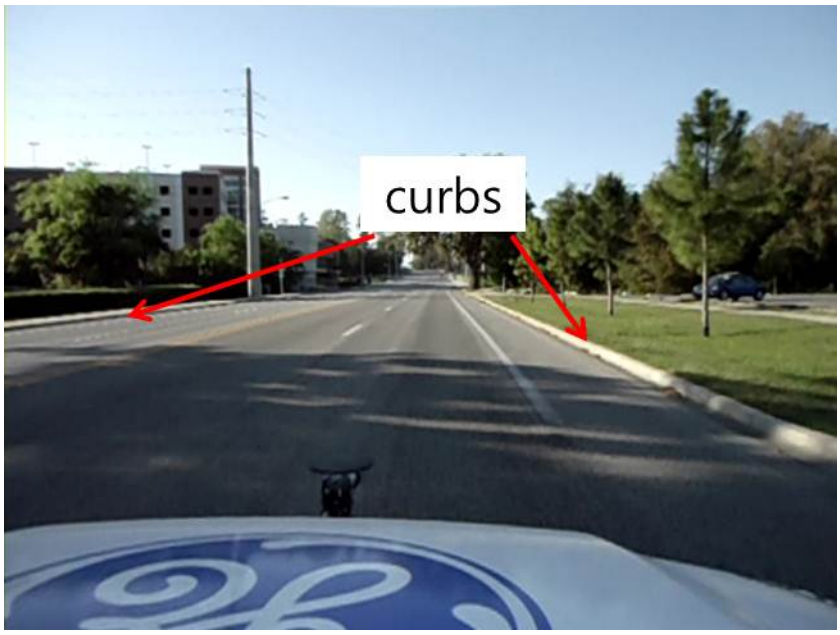
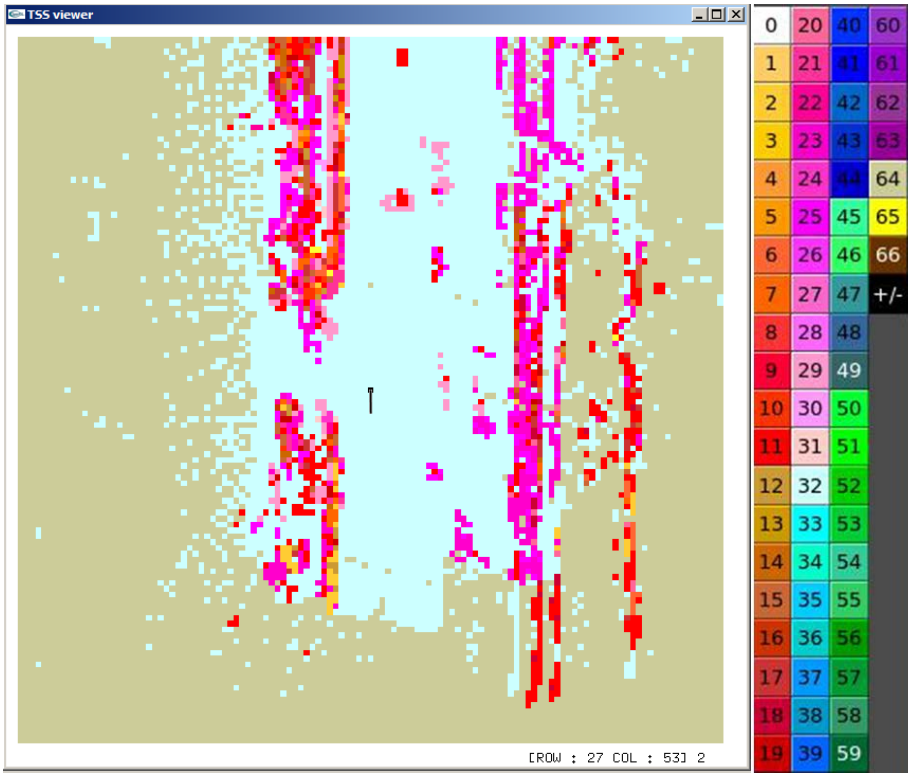


Figure 5-5. Gainesville Raceway pit area for parallel parking behavior on the Google satellite map.



(a)



(b)

Figure 5-6. (a) The picture taken by JihyunYoon from the Gale Lemerand road in the University of Florida campus, (b) Edge result map and traversability value

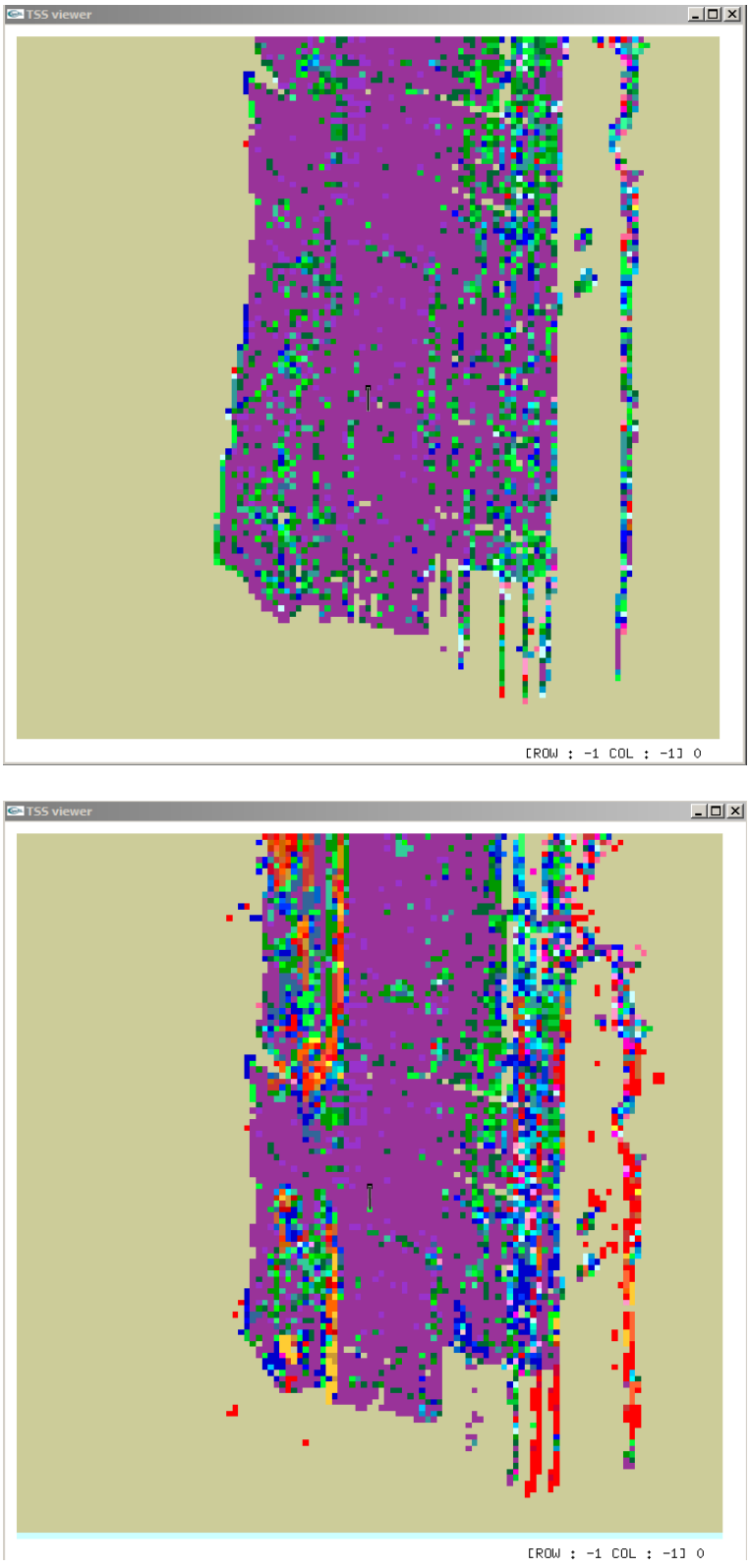


Figure 5-7. Grid map result comparison

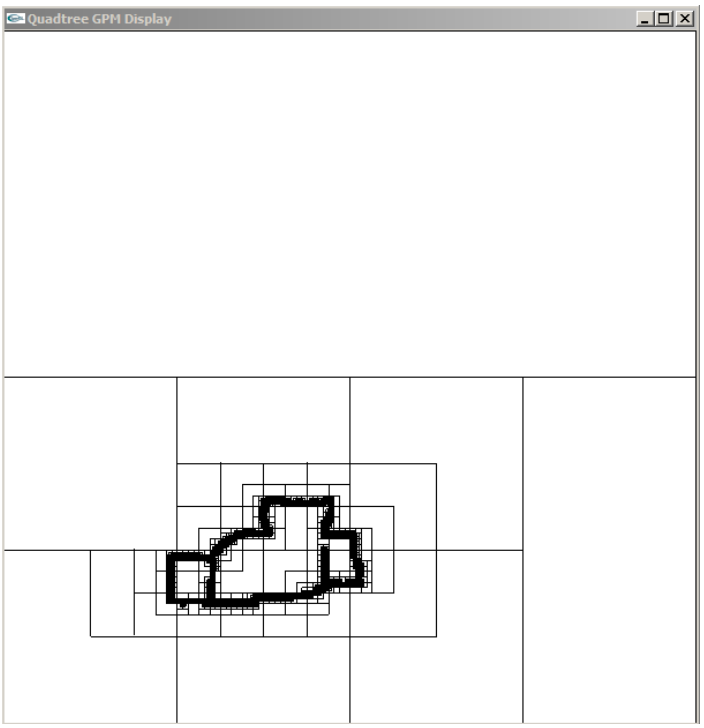


Figure 5-8. Full view of final results of Sensor Knowledge Store

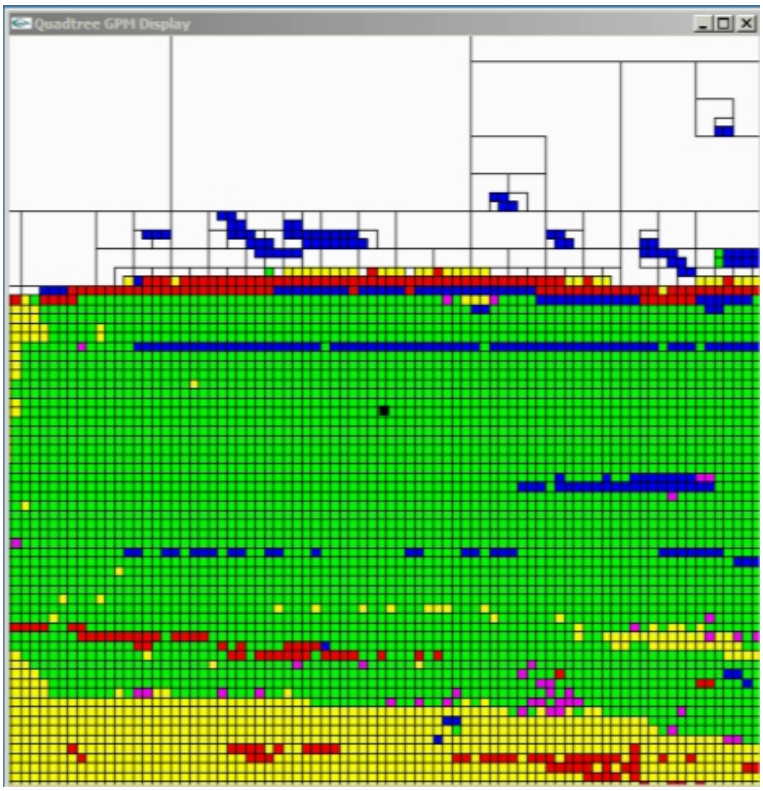


Figure 5-9. Local view of final results of Sensor Knowledge Store in area2

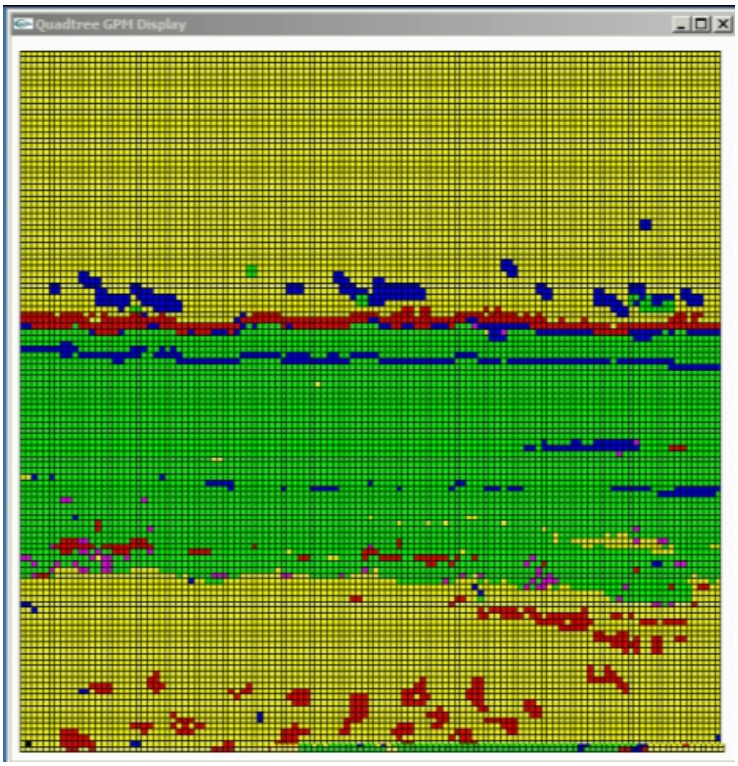
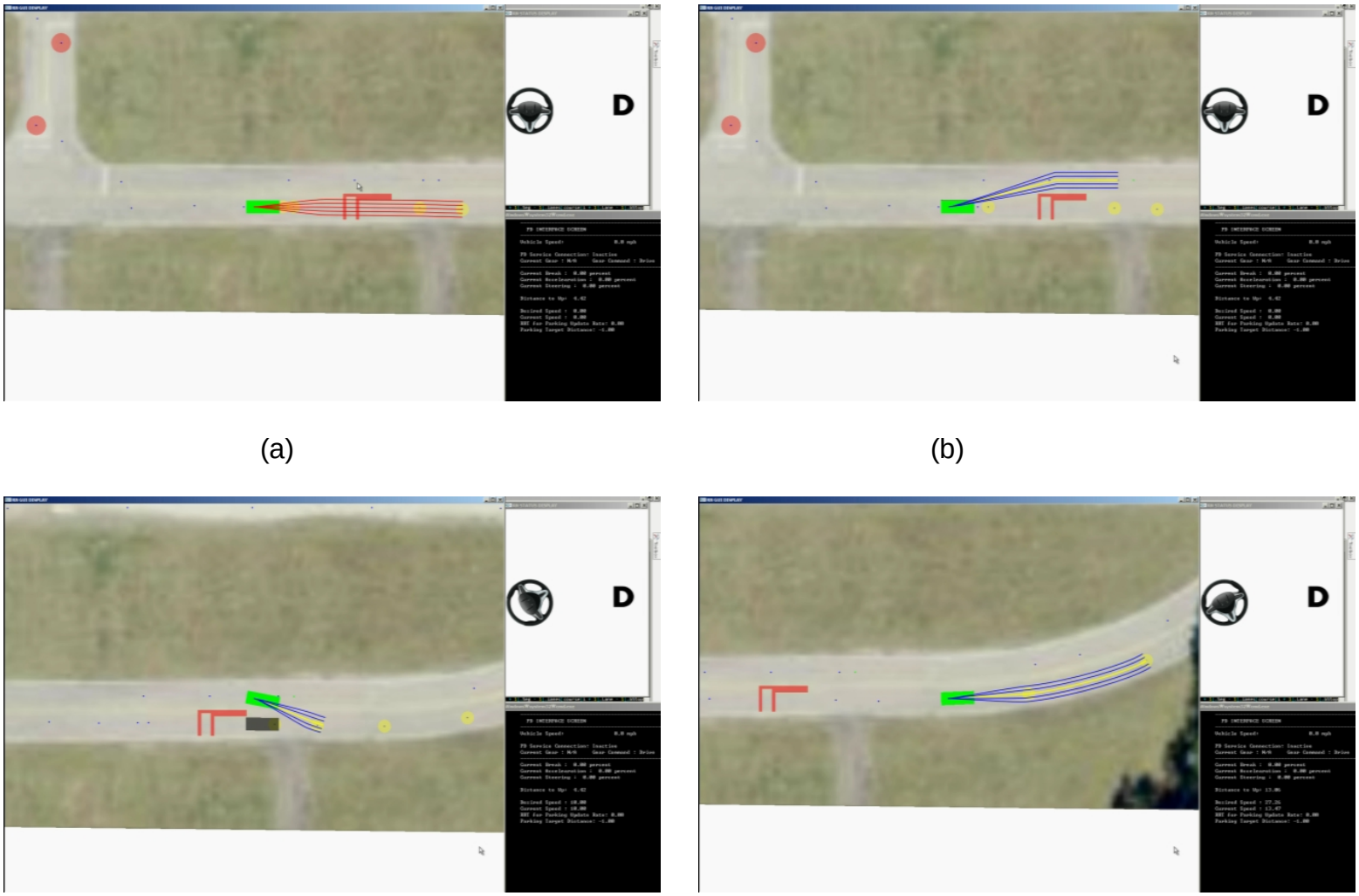


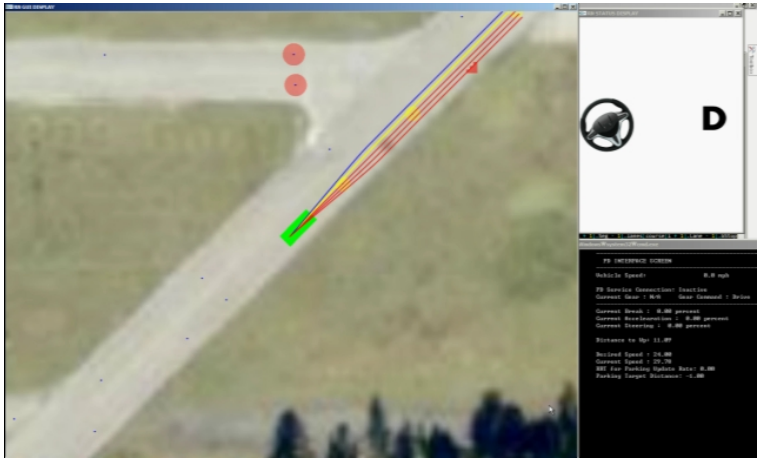
Figure 5-10. Extracted array from area of Figure 5-9



(a) (b)

(c) (d)

Figure 5-11. Lane change behavior for single lane. a) The vehicle stops in safe distance, b) Path generation to the next lane, c) Returning to the original lane, d) Returning to roadway navigation behavior.

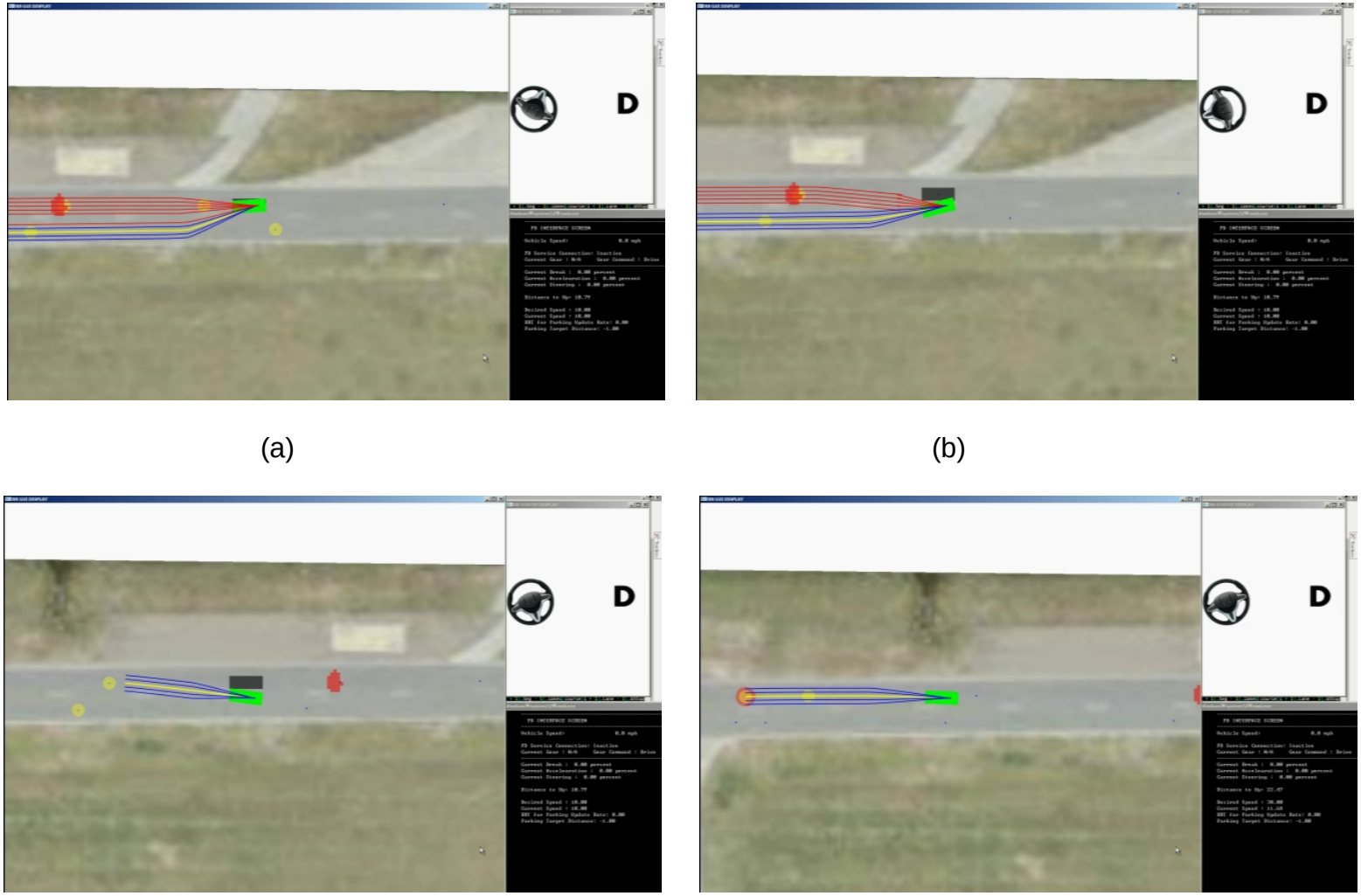


(a)

(b)

(c)

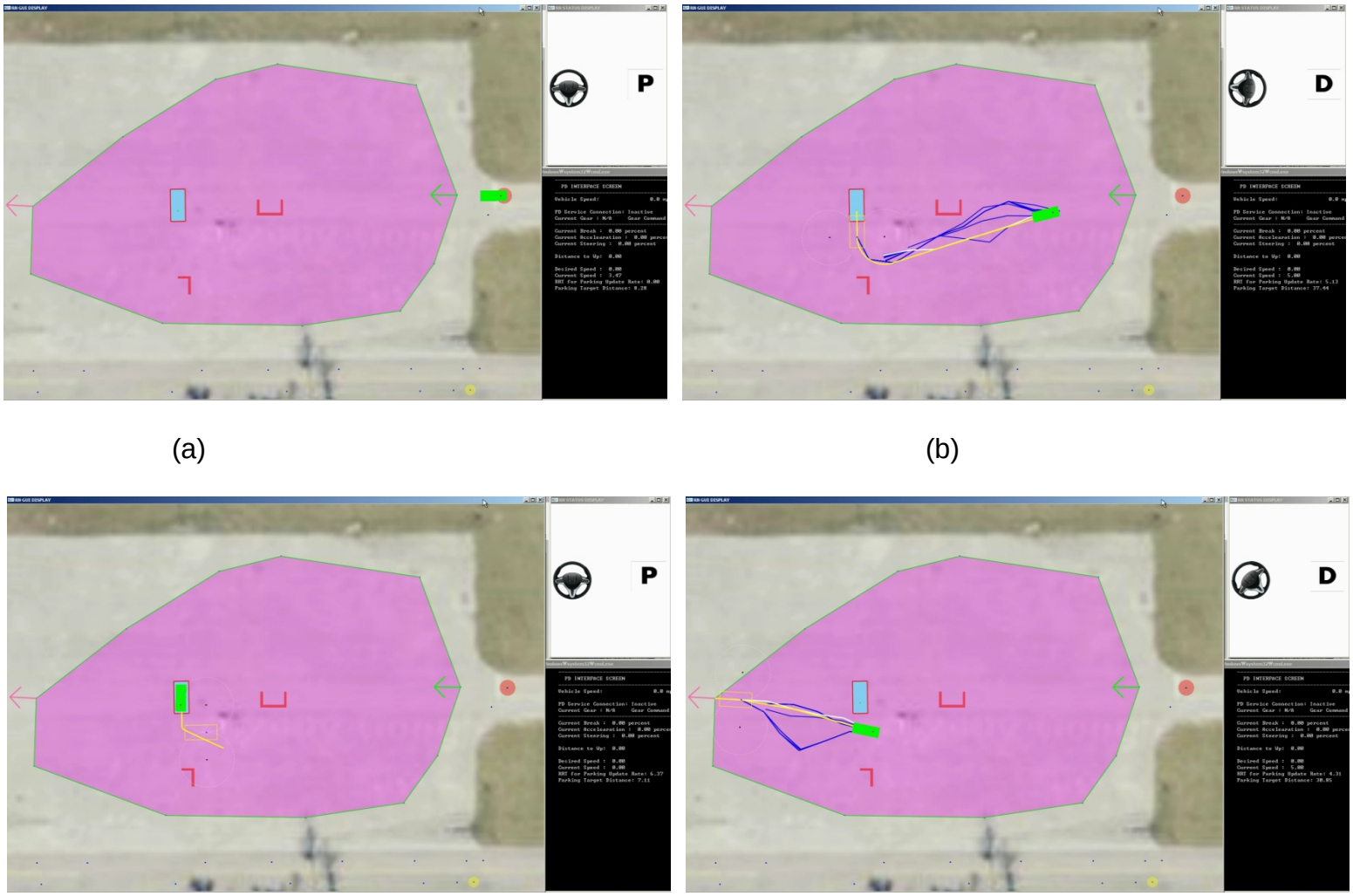
Figure 5-12. Obstacle avoidance in a lane. a) Finding optimal path to avoid the obstacle, b) Avoiding the obstacle by following the optimal path, c) Returning to roadway navigation behavior.



(a) (b)

(c) (d)

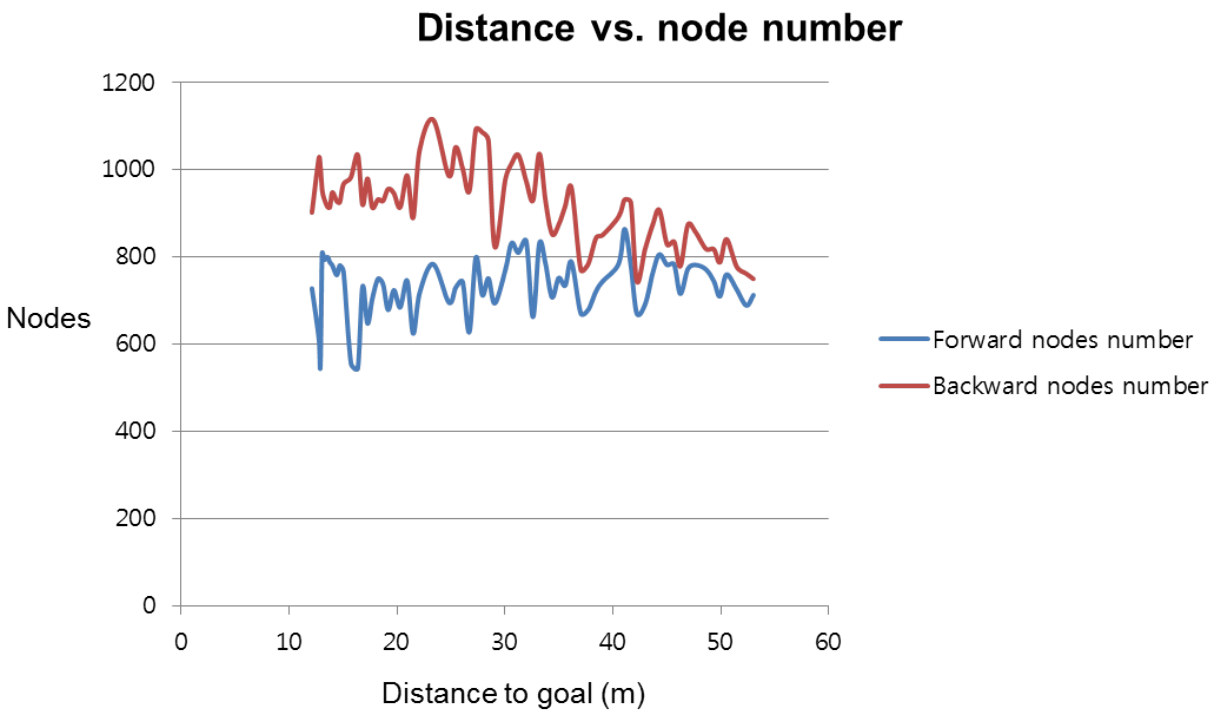
Figure 5-13. Lane change behavior for multiple lanes. a) Finding the optimal path to the next lane without stop, b) Moving to the next lane by following the optimal path, c) Returning the original lane, d) Returning to roadway navigation behavior.



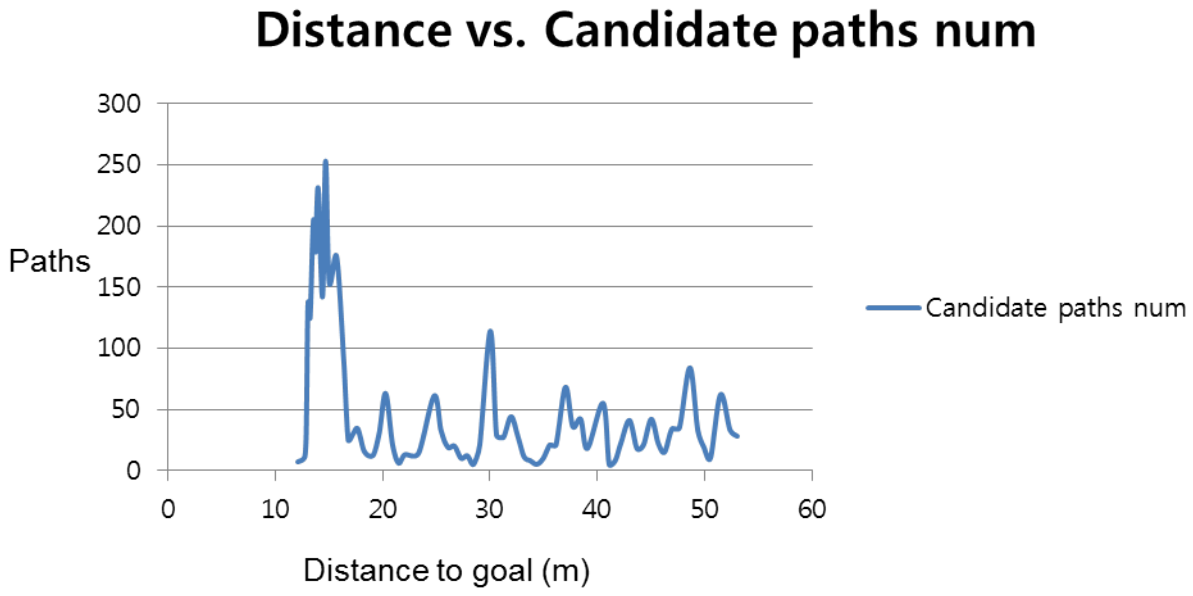
(a) (b)

(c) (d)

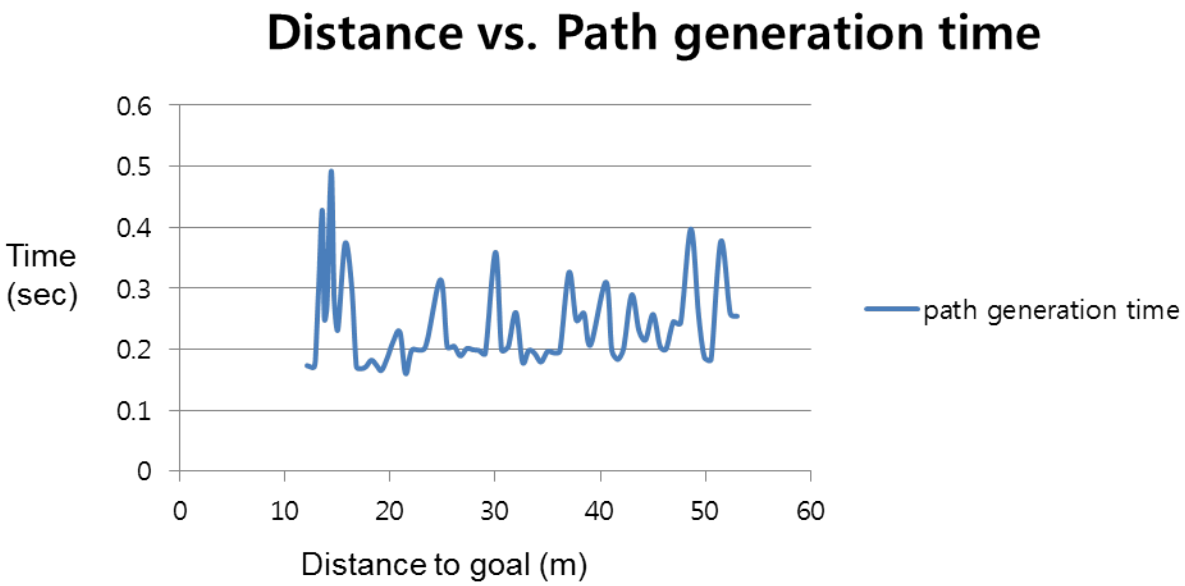
Figure 5-14. Typical parking behavior. a) Initial position for the typical parking problem, b) Approaching to the parking lot, c) Completed parking, d) Approaching to the exit of the parking area.


Distance vs. node number
Nodes
1200
1000
800
600
400
200
0
0
10
20
30
40
50
60
Distance to goal (m)
Forward nodes number
Backward nodes number

(a)

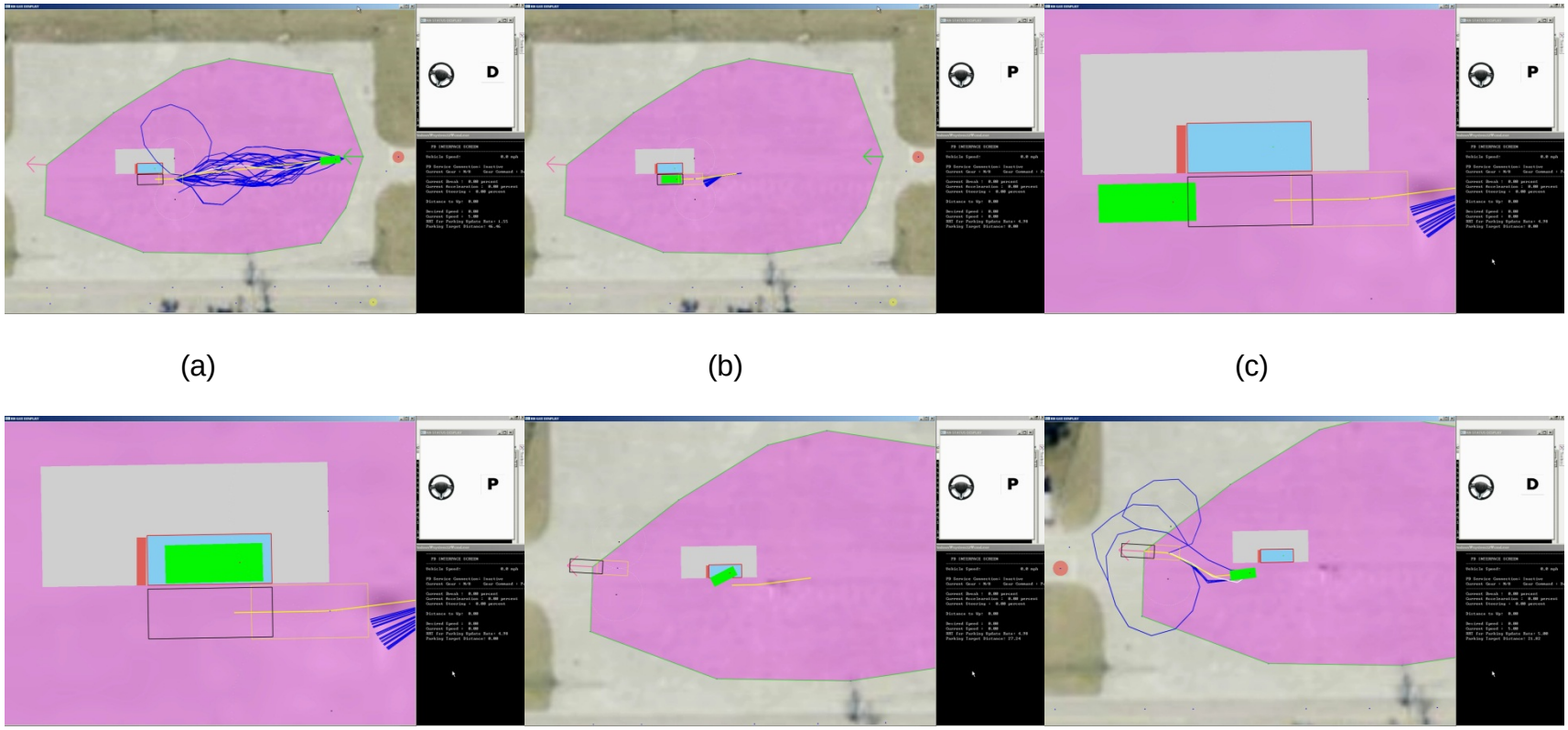

Distance vs. Candidate paths num
Paths
300
250
200
150
100
50
0
0
10
20
30
40
50
60
Distance to goal (m)
Candidate paths num

(b)



(c)

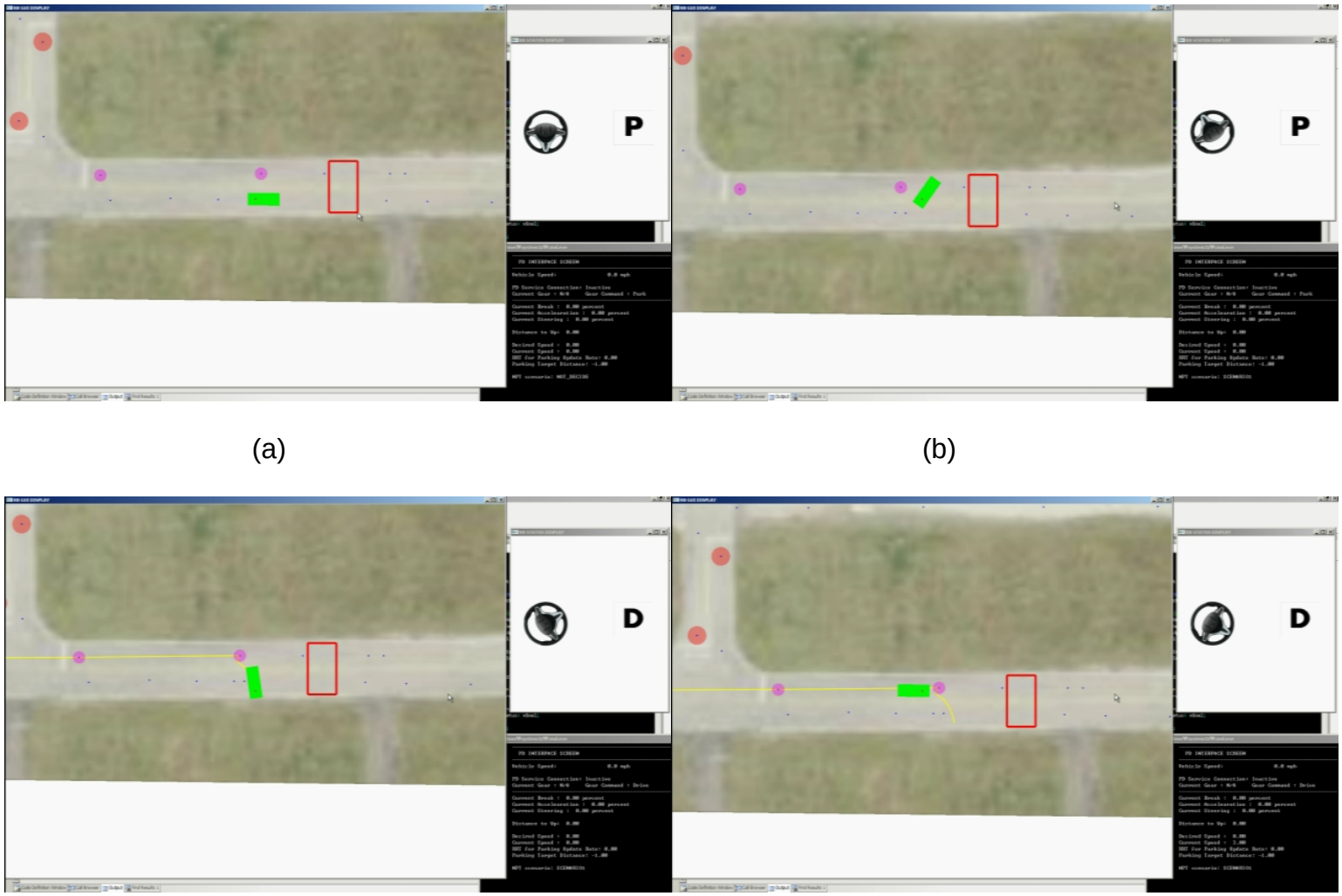
Figure 5-15. Parametric results of parking behavior using RRT



(a) (b) (c)

(d) (e) (f)

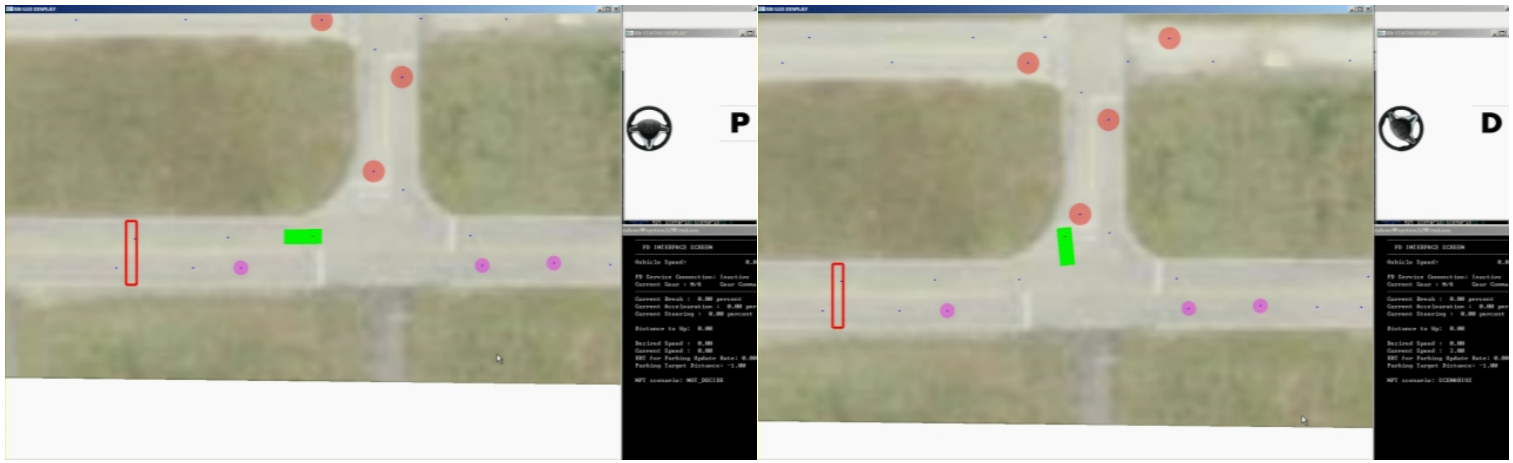
Figure 5-16. Parallel parking behavior



(a) (b)

(c) (d)

Figure 5-17. An n-point turn for case 1. a) Stop in safe distance, b) The first movement for n-point turn, c) Path generation to change driving direction, d) Returning to roadway navigation behavior.



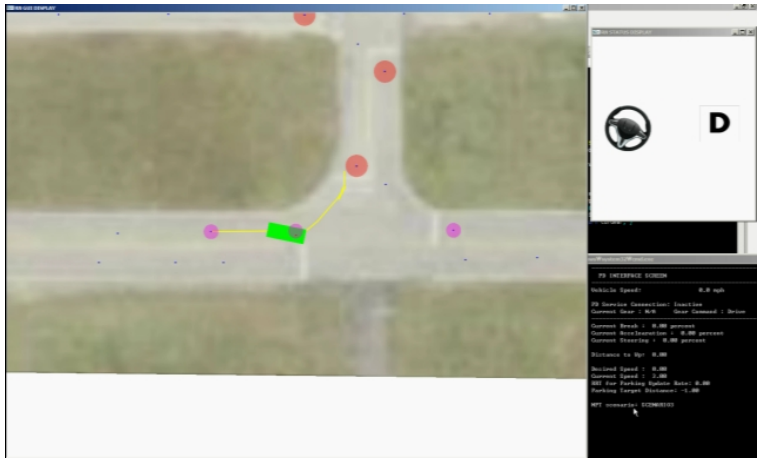
(a) (b)

(c)

Figure 5-18. An n-point turn for case 2. a) Stop in safe distance, b) Changing the vehicle direction by driving backward, c) Returning to roadway navigation behavior.

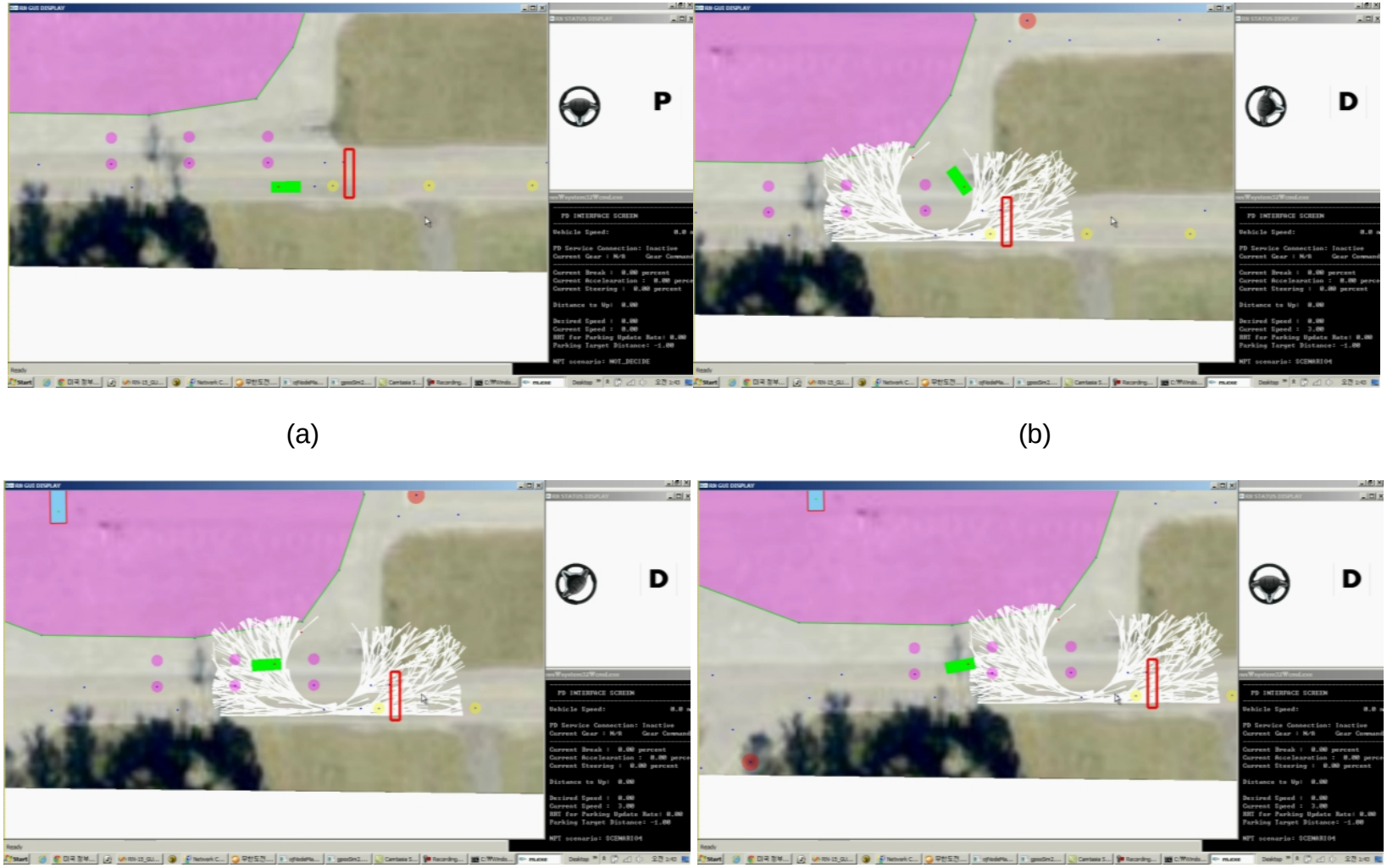
(a)

(b)



(c)

Figure 5-19. An n-point turn for case 3. a) Stop in safe distance, b) Changing the vehicle direction by driving backward, c) Returning to roadway navigation behavior.



(a) (b)

(c) (d)

Figure 5-20. An n-point turn for case 4.) Stop in safe distance, b) Changing the vehicle direction by driving forward, c) Approaching to the one of goal points d) Returning to roadway navigation behavior

Table 5-1. Road Boundaries Detector Uprate.

Situation	Average update rate (Hz)
Desired	40.00
Without sending message	38.52
With sending only grid map message	37.56
With sending edge grid map and height map	36.52
With sending all messages	33.85

Table 5-2. Test results for road boundary detection of area1.

LADAR name	Terrain	Negative	Driver VF	Passenger VF
Scan lines number	1000	1000	1000	1000
Edges lines number	896	847	512	683
Peak detector (%)	82.45	79.56	65.35	71.43
Grid method (%)	76.53	73.23	55.29	65.27
Slope method (%)	92.58	91.25	81.39	85.92

Table 5-3. Test results for road boundary detection of area2.

LADAR name	Terrain	Negative	Driver VF	Passenger VF
Scan lines number	1000	1000	1000	1000
Edges lines number	952	939	629	735
Peak detector (%)	88.50	77.92	62.19	75.20
Grid method (%)	79.37	71.51	62.21	69.31
Slope method (%)	95.81	93.57	84.70	88.20

Table 5-4. Test results for road boundary detection of area3.

LADAR name	Terrain	Negative	Driver VF	Passenger VF
Scan lines number	1000	1000	1000	1000
Edges lines number	984	977	628	803
Peak detector (%)	89.22	79.56	65.35	71.43
Grid method (%)	71.29	69.36	49.31	75.41
Slope method (%)	96.88	94.30	80.91	88.72

Table 5-5. Test results for road boundary detection of area4.

LADAR name	Terrain	Negative	Driver VF	Passenger VF
Scan lines number	1000	1000	1000	1000
Edges lines number	909	851	617	803
Peak detector (%)	79.73	71.37	60.21	69.77
Grid method (%)	71.43	79.88	52.21	59.97
Slope method (%)	89.74	87.50	80.21	83.01

Table 5-6. Test results for road boundary detection of area5.

LADAR name	Terrain	Negative	Driver VF	Passenger VF
Scan lines number	1000	1000	1000	1000
Edges lines number	979	925	385	671
Peak detector (%)	87.40	82.87	61.09	75.98
Grid method (%)	82.50	78.09	59.44	71.24
Slope method (%)	93.25	90.47	80.19	88.26

Table 5-7. Test results for road boundary detection of area6.

LADAR name	Terrain	Negative	Driver VF	Passenger VF
Scan lines number	1000	1000	1000	1000
Edges lines number	922	891	698	776
Peak detector (%)	88.53	80.92	59.31	70.22
Grid method (%)	80.29	77.93	55.29	65.27
Slope method (%)	95.37	88.39	76.58	81.96

Table 5-8. Test result for Sensor Knowledge Store

Name	Value
Component update rate (Hz)	39.75
Quadtree building rate (Hz)	19.85
Driving distance (kilometers)	9.27
Average vehicle speed (mph)	20
Tree height	14
Tree range from center (m)	4096 X 4096
Global Center of Quadtree X (m)	369584.000094
Global Center of Quadtree Y (m)	3282071.391349
Covered area size (m^2)	426190.00
Total number of nodes	2290450
Total number of leaf nodes	1704760
Memory usage (Mb)	39.318

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

This dissertation describes new and novel methods for currently existing and newly generated components to support autonomous driving. This final chapter seeks to continue the discussion of this new method by first presenting several areas of potential future work that arose during the development and testing of the new methods. It then draws conclusions from the theory and implementation discussed in Chapters 3 and 4 and from the test results of the validation process described in Chapter 5. It also outlines the main contributions of the proposed methods to the field of robotics.

Future Work

The results described in Chapter 5 show that these approaches can provide better sensing performance, store huge amounts of information effectively, and give more options to the vehicle for path planning, but there is still room to improve the solutions to problems discussed in this dissertation.

For the road boundaries detection, a better and more precise sensing method is necessary. The test in this dissertation was performed at 20 mph, so this update rate was enough to sense the environment. The vehicle will need to run at higher speeds in future, however, so the update rate will have to be higher to provide sufficient information about the environment. In addition, the test results were collected and analyzed for artificial road boundaries, but there are various types of boundaries, so the test shall be performed and verified with those types of boundaries.

Since its range of application is so wide, this Sensor Knowledge Store algorithm is quite useful, although there are still several areas that need to improve. The user may want to save more kinds of data types. For this reason, the algorithm needs to store the

information using more efficient data structures. As the user wants to save more kinds of data types, the future memory should be larger than the current size, so the developers must consider better storage and use it effectively. The last point for improvement is extraction speed. Extraction is the necessary step to send information for necessary areas in proper size, but the process needs a slightly longer time than when using a general grid map since it has to find corresponding nodes one by one. So the developer must find a way to extract the array from the tree directly. Some of these can be addressed with more efficient programming techniques and others with faster hardware and greater storage capacity.

Immediate future work will deal with testing the algorithm on a real vehicle in cases where moving obstacles are introduced into the environment. The methods proposed in this dissertation were vector-base and the method presented is an A* algorithm that is grid-based. There is no solution yet to the question of which one is better, so a new method that combines two algorithms may be a good solution for the problem. The parking algorithm in this dissertation shows a navigation algorithm that incorporates a rapidly exploring random tree method. To reduce the search time, many constraints were used in the algorithm. The feasibility of this proposed method has been demonstrated via simulation, using performance parameters obtained from an existing autonomous vehicle system.

Conclusions

Sensing and path planning tasks are by any means not trivial, even after decades of research and development. The ability of a robot to sense obstacles and generate an optimal path to move itself from its current state to its goal state without colliding with any objects is very important. Moving obstacles in the robot's environment obviously

makes these problems much more complicated. The 2007 DARPA Urban Challenge, in which unmanned ground vehicles were required to navigate urban environments while interacting with other moving vehicles, including other unmanned vehicles, is good example of the need for moving object detection. The research presented in this dissertation is to provide new and novel approaches to the sensing and path planning methods. The first chapter provides an introduction to the sensing and path planning problems in general, while Chapter 2 recalls previous works that proposed several approaches to solve the problems in this dissertation. This is followed by a discussion of the theory behind the newly presented methods in Chapter 3 and an outline of the current implementation of the method on the Urban NaviGator in Chapter 4. A description of the validation procedure and summary of testing results is then provided in Chapter 5.

The concept of the methods discussed in this dissertation is a particularly novel approach to the presented research, and the test results described in Chapter 5 provided abundant evidence that these new methods are capable of improving autonomous driving not only for unmanned ground vehicles, but for other types of unmanned vehicles and other types of robotic systems. The new methods combined in future research also can provide new potential applications and solutions to existing problems in current robotic systems.

LIST OF REFERENCES

1. H. Durrant-Whyte and T. Bailey. Simultaneous Localization and Mapping (SLAM). *Robotics and Automation Magazine*, 2006.

2. C. Wang, C. Thorpe, and A. Suppe. Ladar-Based Detection and Tracking of Moving Objects from a Ground Vehicle at High Speeds*. IEEE Intelligent Vehicles Symposium (IV2003)*, 2003.

3. M. de Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry : Algorithms and applications.* Springer-Verlag, 2000.

4. *DARPA Urban Challenge Technical Evaluation Criteria*, from http://www.darpa.mil/grandchallenge/rules.asp, 2006.

5. S. Solanki. *Development of Sensor Componenet for Terrain Evaluation and Obstacle Detection for an Unmanned Autonomous Vehicle.* Ph.D Dissertation, University of Florida, 2007.

6. P. Hart, N. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE International Conference on Robotics and Automation*, 1968.

7. J. Leonard and H. Durrant-Whyte. Mobile Robot Localization by Tracking Geometric Beacons*. IEEE Transactions on Robotics and Automation,* pages 376 - 382, 1991

8. D. Cole and P. Newman. Using Laser Range Data for 3D SLAM in Outdoor Environments. *IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

9. R. Kummerle, B. Steder, C. Dornhege, A. Kleiner, G. Grisetti, and W. Burgard. Large Scale Graph-based SLAM using Aerial Images as Prior Information*. Robotics: Science and Systems*, 2009.

10. C. Wang and C. Thorpe. Simultaneous Localization and Mapping with Detection and Tracking of Moving Objects*. IEEE international Conference on Robotics and Automation (ICRA)*, 2002.

11. J. Zhu, N. Zheng, Z. Yuan, Q. Zhang, X. Zhang, and Y. He. A SLAM Algorithm Based on the Central Difference Kalman Filter*. IEEE Intelligent Vehicles Symposium*, 2009

12. X. Yuan, C. Zhao, and Z. Tang. Lidar Scan-matching for Mobile Robot Localization*. Information Technology Journal,* 9: 27-33, 2009.

13. C. Brenner. Vehicle Localization using Landmarks Obtained by a LIDAR Mobile Mapping System*. Paparoditis N., Pierrot-Deseilligny M., Mallet C., Tournaire O. (Eds), IAPRS, Vol. XXXVIII, Part 3A*, 2010.

14. N. Naikal, A. Zakhor, and J. Kua. Image Augmented Laser Scan Matching for Indoor Localization*.* Technical Report No. UCB/EECS-2009-35, University of California, Berkeley, 2009.

15. M. Bosse and R. Zlot. Keypoint Design and Evaluation for Place Recognition in 2D LIDAR Maps*. Robotics and Autonomous Systems, Volume 57,* pages 1211-1224, 2009.

16. M. Bosse and R. Zlot. Place Recognition Using Keypoint Similarities in 2D Lidar Maps*. Experimental Robotics, Springer Tracts in Advanced Robotics*, pages 363-372, 2009.

17. D. Frenkel. *Introduction to Monte Carlo Methods.* NIC Series, Vol. 23, page 29-60, 2004.

18. D. Fox, W. Burgard, F. Dellaert, and S. Thrun, Monte Carlo Localization: Efficient Position Estimation for Mobile Robots*. National Conference on Artificial Intelligence (AAAI),* 1999.

19. O. Wulf and B. Wagner, Using 3D Data for Monte Carlo Localization in Complex Indoor Envronments*. European Conference on Mobile Robots (ECMR)*, 2005.

20. R. Kummerle, R. Triebel, P. Pfaff, and W. Burgard. Monte Carlo Localization in Outdoor Terrains using Multilevel Surface Maps*. Journal of Field Robotics,* Vol. 25, Issue 6-7, pages 346-359, 2008.

21. M. Hebert, and N. Vandapel. Terrain Classification Technique from LADAR Data for Autonomous Navigation*. Collaborative Technology Alliances conference*, 2003.

22. C. Yu, and X. Yua. Terrain Classification for Autonomous Navigation Using Ladar Sensing*. Information Science and Engineering (ICISE)*, 2009.

23. M. Wang and Y. Tseng., LiDAR Data Segmentation and Classification Based on Octree Structure*. International Archives of Photogrammetry and Remote Sensing,* ISSN 1682-1750, Vol.XXXV, part B3, 2004.

24. J. Bauer, K. Karner, K. Schindler, A. Klaus, and C. Zach, Segmentation of Building Models from Dense 3D point-clouds*. 27th Workshop of the Austrian Association for Pattern Recognition*, 2003.

25. F. Moosmann, O. Pink, and C. Stiller. Segmentation of 3D Lidar Data in non-flat Urban Environments using a Local Convexity Criterion*. IEEE Intelligent Vehicles Symposium*, 2009.

26. D. Belton and D. L., Classification and Segmentation of Terrestrial Laser Scanner Point Clouds Using Local Variance Information*. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Vol. XXXVI, Part 5, pages 44-49 2006.

27. N. Chehata, N. David, and F. Bretar. LIDAR Data Classification using Hierarchical K-mean clustering*. International Society for Photogrammetry and Remote Sensing (ISPRS),* 2008.

28. A. Golovinskiy, T. Funkhouser, and V. Kim, Shape-based Recognition of 3D Point Clouds in Urban Environments*. International Conference on Computer Vision (ICCV)*, 2009.

29. W. Zhang. LIDAR-Based Road and Road-Edge Detection. *IEEE Intelligent Vehicles Symposium*, 2010.

30. F.Neuhaus, D. Dillenberger, J.Pellenz, and D. Paulus. Terrain Drivability Analysis in 3D Laser Range Data for Autonomous Robot Navigation in Unstructured Environments*. Emerging Technologies & Factory Automation*, 2009.

31. W. Wijesoma, K. Kodagoda, and A. Balasuriya, Road-Boundary Detection and Tracking Using Ladar Sensing*. IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION,* VOL. 20, NO. 3, 2004.

32. R. Manduchi. Obstacle Detection and Terrain Classification for Autonomous Off-Road Navigation*. Autonomous Robots 18,* pages 81–102, 2005.

33. M. Hebert, N. Vandapel, S. Keller, and R. Donamukkala. Evaluation and Comparison of Terrain Classification Techniques from LADAR Data for Autonomous Navigation*. Army Science Conference*, 2002.

34. J. Sparbert, K. Dietmayer, and D. Streller. Lane Detection and Street Type Classification using Laser Range Images*. IEEE Intelligent Transportation Systems Conference*, 2001.

35. D. Stavens and S. Thrun. A Self-Supervised Terrain Roughness Estimator for Off-Road Autonomous Driving*. Conference on Uncertainty in AI*, 2006.

36. N. Heckman, J. Lalonde, N. Vandapel, and M. Hebert. Potential Negative Obstacle Detection by Occlusion Labeling*. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.

37. J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, and M. Walter, A Perception Driven Autonomous Urban Vehicle*. Journal of Field Robotics*, 2008.

38. A. Bargeton and F. Nashashibi. Laser-based Vehicles Tracking and Classification using Occlusion Reasoning and Confidence Estimation*. IEEE Intelligent Vehicles Symposium*, 2008.

39. U. Ramer. An Iterative Procedure for the Polygonal Approximation of Plane Curves*. Computer Graphics and Image Processing,* 1(3): 244–256, 1972.

40. C. Stiller, J. Hipp, C. Rossig, and A. Ewald. Multisensor Obstacle Detection and Tracking*. Image and Vision Computing 18,* pages 389–396, 2000.

41. J. Hancock, M. Herbert, and C. Thorpe. Laser Intensity-Based Obstacle Detection*. IEEE/RSJ International Conference On Intelligent Robotic Systems (IROS '98)*, 1998.

42. C. Laugier, I. Paromtchik, M. Perrollaz, M. Yong, A. Nègre, J. Yoder, and C. Tay. ArosDyn: Robust Analysis of Dynamic Scenes by means of Bayesian Fusion of Sensor Data - Application to the Safety of Car Driving*. ICRA10 Workshop on Robotics and Intelligent Transportation System*, 2010.

43. C. Premebida, G. Monteiro, U. Nunes, and P. Peixoto. A Lidar and Vision-based Approach for Pedestrian and Vehicle Detection and Tracking. *IEEE Intelligent Transportation Systems Conference*, 2007.

44. X. Liu, Z. Zhang, LIDAR Data Reduction for Efficient and High Quality DEM Generation*. XXI Congress of the International Society of Photogrammetry and Remote Sensing (ISPRS2008)*, 2008.

45. T. Peucker, R. Fowler, J. Little, The Triangulated Irregular Network*. American Society of Photogrammetry: Digital Terrain Models (DTM) Symposium*, 1978.

46. Q. Clevis, G. Tucker, S. Lancaster, A. Desitter, N. Gasparini, and G. Locke, A Simple Algorithm for the Mapping of TIN Data onto a Static Grid: Applied to the Stratigraphic Simulation of River Meander Deposits*. Computers and Geosciences*, Volume 32, Issue 6, pages 749-766, 2006.

47. A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching*. ACM SIGMOD International Conference on Management of Data,* pages 47-57, 1984.

48. S. Ravada, M. Horhammer, and B. Kazar, Oracle Spatial Point Cloud: Storage, Loading, and Visualization*. Oracle*, 2010.

49. B. Schön, M. Bertolotto, ,D. Laefer, and S. Morrish. Storage, Manipulation, and Visualization of LiDAR Data*. International Society of Photogrammetry and Remote Sensing*, 2009.

50. J. Bentley and R. Finkel. Quad Trees: A Data Structure for Retrieval on Composite Keys*. Acta Informatica 4*, (1): 1–9., 1974.

51. W. Tobler and Z. Chen *A*. Quadtree for Global Information Storage*. Geographical Analysis*, 18(4): 360-371, 1986.

52. J. Li, H. Fan, H. Ma, and S. Goto. Determination of Large-Scale Digital Elevation Model in Wooded Area with Ariborne LIDAR Data by Applying Adaptive Quadtree-based Iterative Filtering Method*. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Science,* Volume XXXVIII, Part 8, 2010.

53. J. Lee, R. Mottaghi, C. Pippin and T. Balch. Graph-based Planning Using Local Information for Unknown Outdoor Environments*. IEEE International Conference Robotics and Automation(ICRA '09)*, 2009.

54. M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime Dynamic A*: An Anytime, Replanning Algorithm*. International Conference on Automated Planning and Scheduling (ICAPS)*, 2005.

55. Dave Ferguson and A. Stenz, Field D* : An Interpolation-based Path Planner*. International Symposium on Robotics Research (ISRR)*, 2005.

56. A. Yahja, A. Stenz, S. Singh, and B. Brumitt. Framed-Quadtree Path Planning for Mobile Robots Operating in Sparse Environments*. IEEE Conference on Robotics and Automation (ICRA)*, 1998.

57. L. Kavraki, P. Svestka, J. Latombe, and M. Overmars. Probabilistic Roadmaps for Path Planning in High-dimensional Configuration Spaces*. IEEE Transactions on Robotics and Automation*, 12 (4): 566–580, 1996.

58. L. Jaillet, J. Cortés, and T. Siméon. Sampling-Based Path Planning on Configuration-Space Costmaps*. Robotics, IEEE Transactions*, 2010.

59. L. Kavraki, M. Kolountzakis, and J. Latombe, Analysis of Probabilistic Roadmaps for Path Planning*. IEEE Transactions on Robotics and Automation,* Volume 14, Issue 1, Number 1, pages166-171*,* 1998.

60. R. Bohlin and L. Kavraki. Path Planning using Lazy PRM*. Robotics and Automation*, 2000.

61. O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*. International Journal of Robotic Research,* Vol.5, No.1, 1986.

62. J. Barraquand, B. Langlois, and J. Latombe. Numerical Potential Field Techniques for Robot Path Planning*. IEEE Transactions on Systems, Man and Cybernetics,* Volume 22, Issue 2, 1992.

63. Y.Hwang and N. Ahuja. A Potential Field Approach to Path Planning*. IEEE Transaction on Robotics and Automation,* VOL. 8, NO. 1, 1992.

64. J. Borenstein and Y.Koren. The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots*. IEEE Journal of Robotics and Automation,* Vol 7, No 3, 1991.

65. A. Maida, S.Golconda, P. Mejia, A. Lakhotia, and C. Cavanaugh. Subgoal-based Local-navigation and Obstacle-avoidance using a Grid-distance Field*. International Journal of Vehicle Autonomous Systems (IJVAS),* pages 122-142, 2006.

66. L. Dubins. On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents*. American Journal of Mathematics,* Vol. 79, No. 3, 1957.

67. R. Takei, R. Tsai, H. Shen, and Y. Landa. A Practical Path-planning Algorithm for a Vehicle with a Constrained Turning Radius: a Hamilton-Jacobi Approach*. American Control Conference*, 2010.

68. L. Han, H.Yashiro, H. Nejad, Q. Do, and S. Mita. Bézier Curve Based Path Planning for Autonomous Vehicle in Urban Environment*. IEEE Intelligent Vehicles Symposium*, 2010.

69. K. Usher. Obstacle Avoidance for a Nonholonomic Vehicle using Occupancy Grids*. Australasian Conference on Robotics & Automation (ACRA)*, 2006.

70. C. Fulgenzi, A. Spalanzani, and C. Laugier. Dynamic Obstacle Avoidance in Uncertain Environment Combining PVOs and Occupancy Grid. *IEEE International Conference on Robotics and Automation*, 2007

71. Ziegler, J., Stiller, C, *Fast Collision Checking for Intelligent Vehicle Motion Planning.* Intelligent Vehicles Symposium (IV), 2010 IEEE.

72. J. Snider. Automatic Steering Methods for Autonomous Automobile Path Tracking. technical report CMU-RI-TR-09-08, Robotics Institute, Carnegie Mellon University, 2009.

73. S. Singh and D. Shin. Position Based Path Tracking for Wheeled Mobile Robots*. IEEE Conference on Intelligent Robot Systems*, pages 386 - 391, 1989.

74. S. Singh and D. Shin. Explicit Path Tracking by Autonomous Vehicles*. Robotica,* Vol. 10, pages 539 - 554, 1992.

75. D. Gu and H. Hu. A Stabilization Receding Horizon Regulator for Nonholonomic Mobile Robots*. IEEE Transactions on Robotics,* Journal Volume 21, Issue 5, pages 1022-1028, 2005.

76. E. Thorn. *Autonomous Motion Planning Using A Predictive Temporal Method.* Ph.D. Disseratation. University of Florida, 2009.

77. S. LaValle. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University, 1998.

78. A. Savitzky and M. Golay. Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Analytical Chemistry*, 36 (8): 1627-1639, 1964.

79. T. Galluzo. *Simultaneous Planning and Control for Autonomous Ground Vehicles.* Ph.D. Dissertation. University of Florida, 2006.

BIOGRAPHICAL SKETCH

Jihyun Yoon was born and raised in Jinhae, Kyungnam, South Korea. He received his B.S. degree in automotive engineering from the Kookmin University in Seoul, South Korea in February 2006. He is currently working as a Graduate Research Assistant and completing a doctoral degree at the Center for Intelligent Machines and Robotics (CIMAR) at the University of Florida. His research focuses on unmanned ground vehicle LADAR sensing and path planning. He plans to continue his career as a research engineer at Samsung Electronics in South Korea.